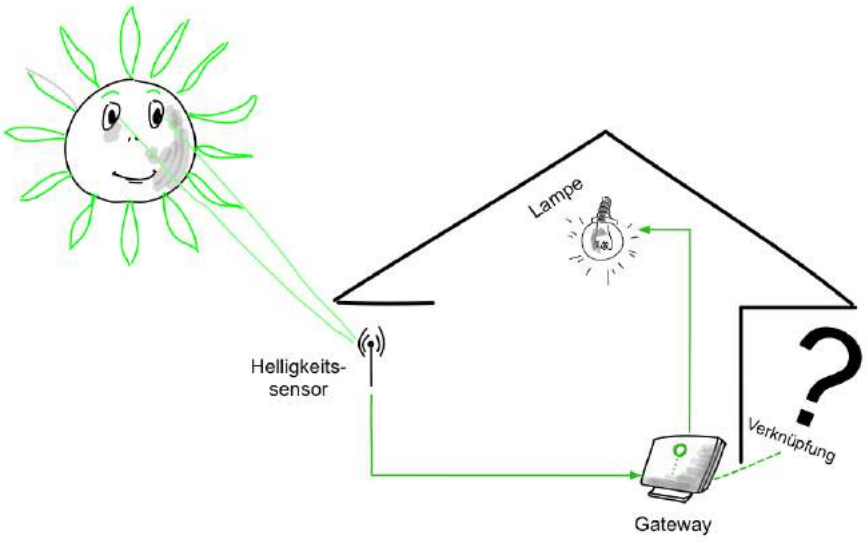
Kluges Haus, was tun?

Im folgenden wollen wir zeigen, wie sich Smart-Home-Devices selbstlernend über den *Xonbot* steuern lassen und warum das so cool ist.

Fangen wir an. Die Beleuchtung im Wohnzimmer soll immer dann automatisch angehen, wenn es Draußen dunkel wird. Technisch im Smart Home kein Problem: Helligkeitssensor und Wohnzimmerlampe sind über Gateway vernetzt und die Daten des Sensors können zum Ein- und Ausschalten der Lampe benutzt werden. Die Frage ist nur: Was ist dunkel?

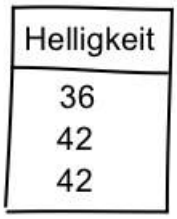


Einfachster Ansatz ist die Festlegung, dass wenn die Helligkeit kleiner 50% ist, die Lampe einzuschalten, sonst auszuschalten. In Regelform:

$$\text{if (Helligkeit} < 50)$$
$$\text{then Lampe} := \text{ein}$$
$$\text{else Lampe} := \text{aus}$$

Diese Regel lässt sich leicht standardmäßig definieren und ausliefern. Leider empfindet Steve Meyer - dies sei unser exemplarischer Hausbesitzer - die Helligkeitsschranke als für sich zu hoch, da es ihm noch zu dunkel ist. Jedoch kann Steve auch nicht genau den Wert nennen, ab dem es ihm hell genug ist. Also probiert er solange aus bis er seine richtige Schranke findet: 40%. Diese trägt er in die Regel ein.

Es geht jedoch auch einfacher. Der Xonbot registriert jeden Tag die Helligkeit, wenn Steve die Lampe einschaltet. In drei Tagen wurden dabei folgende Helligkeiten gemessen:



Daraus berechnet der Xonbot den Mittelwert: 40%. Steve muss hierzu gar nichts tun außer dem Xonbot mitzuteilen, ab wann er sein Schaltverhalten lernen soll und ab wann er es anwenden - also die Lampe selbst schalten - soll. Im Übrigen behält sich Steve natürlich vor, auch dann die Lampe umzuschalten, wenn er es für geraten hält. Der Xonbot lernt und entscheidet also gleichzeitig.

Einschub: Was bringt Echtzeit?

Steve's Sohn Alex ist ein rigider Datenschützer. Zwar postet er alle seine Mahlzeiten auf Twitter, wenn es jedoch um die Frage geht die Werte des Helligkeitssensors zu speichern oder gar ins Internet zu übertragen hört für ihn der Spaß auf! Aber Alex kann geholfen werden.

Wir müssen nämlich die Werte der Helligkeiten gar nicht alle speichern, um den Mittelwert zu berechnen. Stattdessen gibt es einen alternativen Weg: die *inkrementelle* Berechnung des Mittelwerts. Betrachten wir das näher.

Für eine Reihe von Werten x_1, K, x_n lautet die klassische Formel der Mittelwertberechnung

$$\bar{x} = \frac{x_1 + \ldots + x_n}{n}. \qquad \text{(MI)}$$

Sofern nun also ein neuer Wert x_{n+1} hinzukommt, müssen wir die gesamte Rechnung (MI) komplett neu mit allen Werten x_1, K, x_{n+1} wiederholen. Um das zu vermeiden, lässt sich der Mittelwert auch inkrementell berechnen. Dazu benötigen wir lediglich den aktuellen Wert $\bar{x}_n$, also den aktuellen Mittelwert, und die Anzahl der bisherigen Werte *n*. Für einen neuen Wert x_{n+1} lautet dann die Update-Gleichung für den Mittelwert

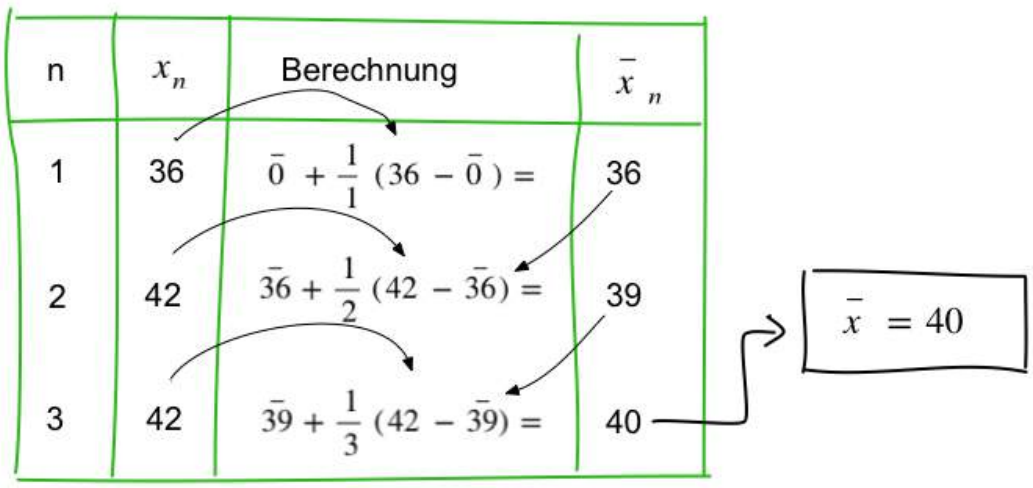
$$\bar{x}_{n+1} = \bar{x}_n + \frac{1}{n+1}(x_{n+1} - \bar{x}_n), \qquad \text{(MR)}$$

wobei wir $\bar{x}_0 = 0$ setzen.

Betrachten wir zur Illustration Steve's Helligkeiten aus dem Eingangsbeispiel, also $x_1 = 36,\ x_2 = 42,\ x_3 = 42$. Wir erhielten als Mittelwert gemäß dem klassischen Ansatz (MI):

$$\bar{x} = \frac{36+42+42}{3} = 40.$$

Gehen wir nun zur rekursive Mittelwertberechnung gemäß (MR) über.



Offensichtlich führt diese zum gleichen Ergebnis. Im Gegensatz zu Ansatz (MI) müssen wir aber nicht alle Werte x_1, K, x_{n+1} speichern und darüber hinaus erfolgt das Update nach Gleichung (MR) weitaus schneller als die Komplettberechnung (MI).

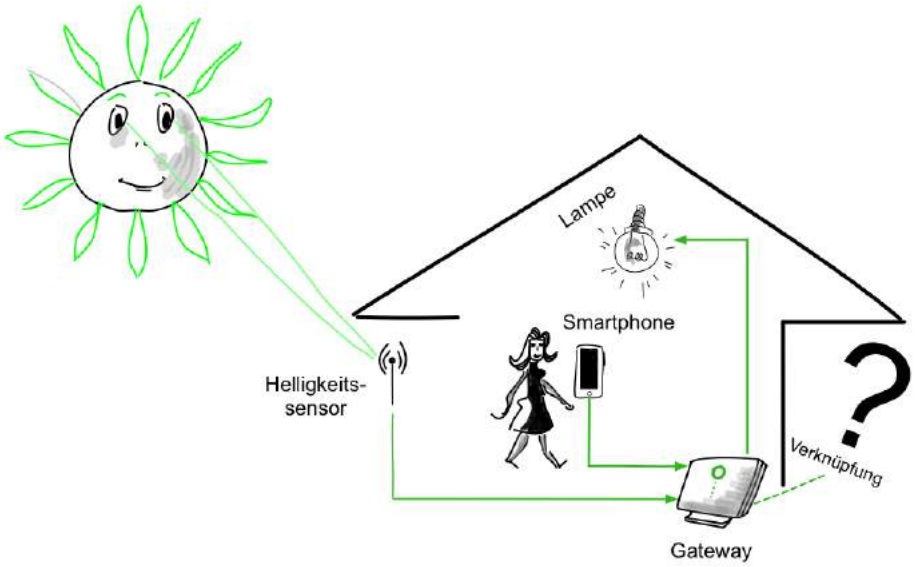
Im vorliegenden Fall ist unser Analysemodell denkbar einfach: der Mittelwert. Die im maschinellen Lernen benutzten Analysemodelle sind weit komplexer und reichen von Entscheidungsbäumen bis zu Neuronalen Netzen ("Deep Learning"). Aber auch hier gibt es die beiden prinzipiellen Ansätze wie in (MI) und (MR): offline und online.

Die meisten derzeit benutzten Verfahren arbeiten offline. Dazu werden alle Daten gespeichert und von Zeit zu Zeit damit die Analysemodelle gelernt. Hingegen lernen inkrementelle Verfahren online, also mit jedem neuen Datensatz. Damit sind die Analysemodelle zum einen immer aktuell. Zum anderen entfällt die Notwendigkeit zur Speicherung der Transaktionsdaten. Derartige Verfahren heißen *Echtzeitanalyseverfahren* (*Realtime Analytics*). Der Xonbot basiert auf Echtzeitanalyse.

Sybille kommt ins Spiel

Steve's Frau Sybille hat es lieber etwas dunkler als Steve. Nach nervigen Tests findet sie heraus, dass ihre Helligkeitsschranke bei 55% liegt. Da sie der Chef im Haus ist, muss bei ihrer Anwesenheit stets ihre Beleuchtungssteuerung gelten. Nehmen wir an, ihre

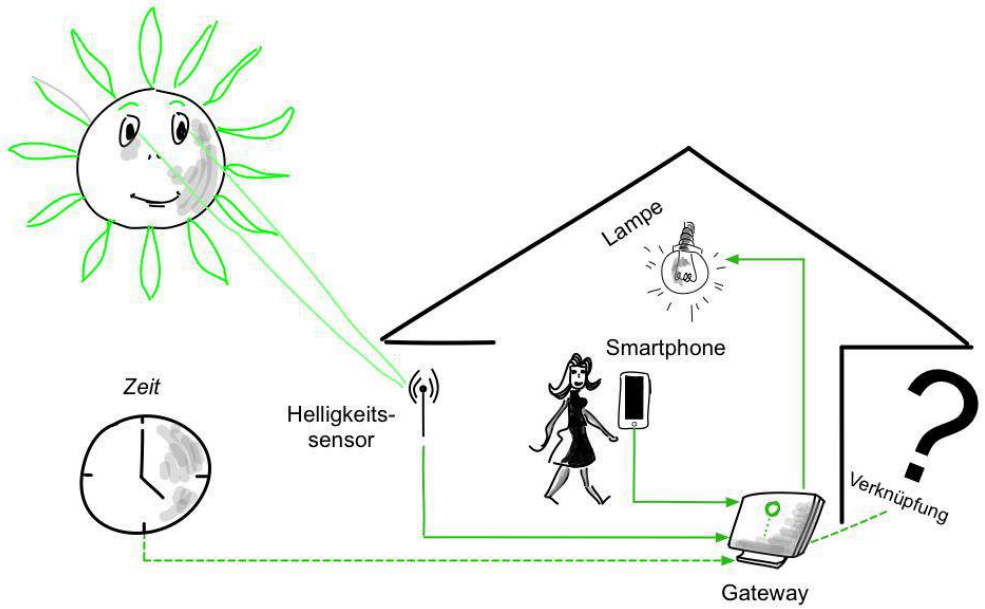
Anwesenheit kann über ihr Smartphone ermittelt werden, welches sie für diesen Zweck freigeschaltet hat.



Dann wird die Regel schon komplexer:

```
if (Sybille) then
{
        if (Helligkeit < 55)
        then Lampe := ein
        else Lampe := aus
} else {
        if (Helligkeit < 40)
        then Lampe := ein
        else Lampe := aus
}
```

Das manuelle Festlegen der Regel wird somit aufwändiger. Je mehr Geräte und Sensoren miteinander zu verknüpfen sind, desto aufwendiger die Vorgabe. Dabei kann der Mensch oft gar nicht mehr alle Zusammenhänge überblicken. So steht Sybille möglicherweise morgens als erste auf, sagen wir um 6 Uhr. Dabei hat sie jedoch nicht immer ihr Smartphone dabei, was bei der manuellen Eingabe der Regel schnell übersehen wird.



Jedoch übersieht der Xonbot nichts und findet:

```
if (Sybille OR (6 Uhr < Zeit < 8 Uhr) ) then
{
        if (Helligkeit < 55)
        then Lampe := ein
        else Lampe := aus
} else {
        if (Helligkeit < 40)
        then Lampe := ein
        else Lampe := aus
}
```

Das Ganze lässt sich beliebig erweitern. Im Ergebnis kristallisieren sich folgende Vorteile des Selbstlernens heraus:

1. **Einfachheit**: Der Xonbot ist einfacher zu bedienen als manuelle Regelvorgabe,
2. **Genauigkeit**: Der Xonbot kann statistisch exakter modellieren als der Mensch,
3. **Korrektheit**: Der Xonbot "übersieht" keine Zusammenhänge.

Bleibt die Frage wie der Xonbot mathematisch funktioniert. Dazu kommen wir später.

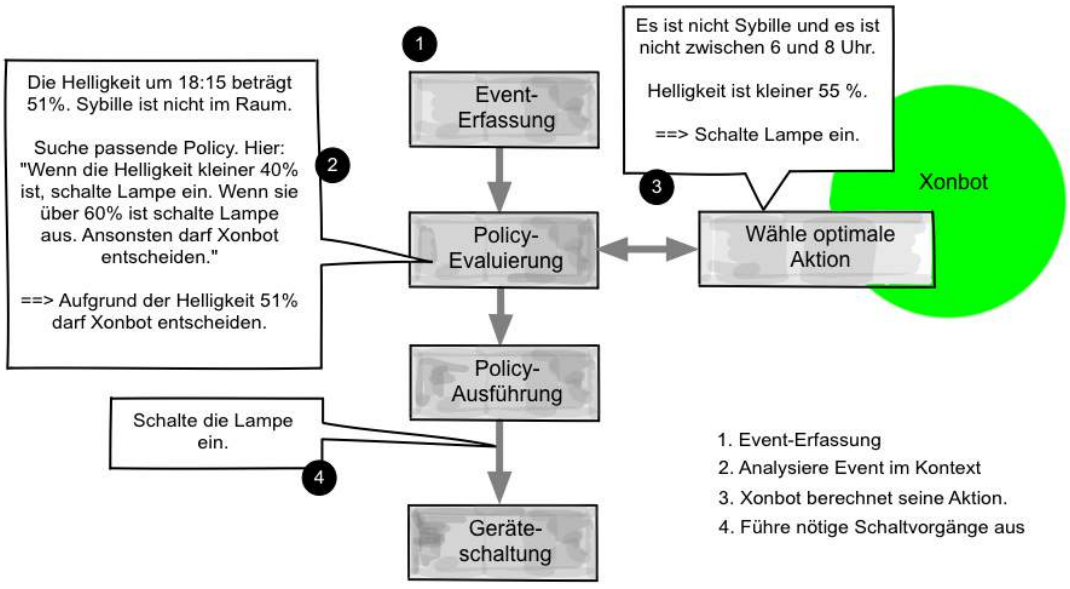
Einschub: Mensch vs. Maschine?

Wir schrieben, dass der Xonbot keine Zusammenhänge übersieht. Das heißt aber nicht, dass er nie irrt! Fehlerquellen gibt es auch für ihn.

Insbesondere wenn zu wenig Daten vorliegen oder falsche, dann kann er auch nicht korrekt lernen. Wenn beispielsweise seit dem Beginn seiner Lernperiode Familie Meyer im Urlaub und die Lampe permanent aus war, dann wird der Xonbot lernen, dass die Lampe unabhängig von der Helligkeit immer ausgeschaltet bleibt. Das ist in einem gewissen Sinne sogar richtig: Für die Urlaubszeit stimmt das Verhalten. Und doch ist es nicht das Gewünschte.

Daher ist es sinnvoll Xonbot auf die Sprünge zu helfen bzw. krasse Fehlentscheidungen durch die Maschine zu verhindern. Dazu müssen wir ihm Vorgaben machen - gewissermaßen seinen Handlungsrahmen festlegen. Diese Vorgaben bezeichnen wir auch als *Constraints* - also Beschränkungen. Üblich ist auch der Begriff *Policy*, denn er macht deutlich, dass die Vorgaben nicht nur Beschränkungen, sondern auch Handlungsanweisungen sind. Die Policies werden in der Praxis meist durch Regeln ausgedrückt.

Kurz gesagt: Der Mensch gibt über Policies das von ihm gewünschte Verhalten vor. Im Rahmen dieser Policies darf der Xonbot tätig werden. Für unser Beispiel der Lampensteuerung kann die Policy z.B. lauten: "Wenn die Helligkeit kleiner 40% ist, schalte die Lampe immer ein. Wenn sie über 60% ist, schalte die Lampe immer aus." Dazwischen ist nichts vorgegeben - der Xonbot darf aktiv werden.



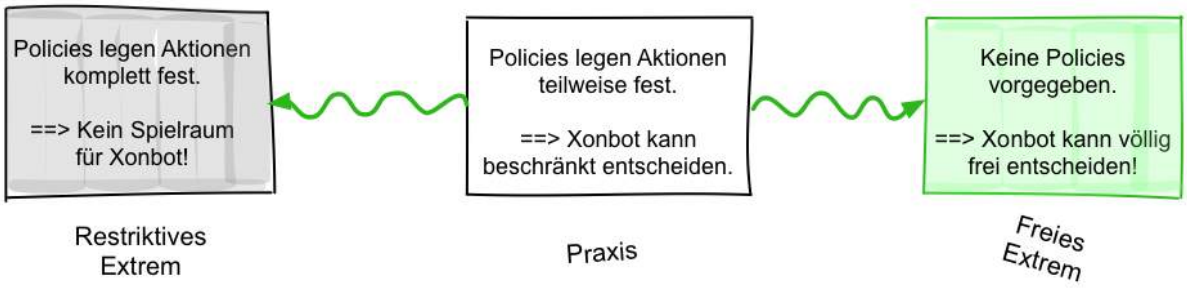
Wo kommen nun diese mystischen Policies eigentlich her? Nun, sie sind schon da! Die meisten Anbieter von Smart-Home-fähigen Geräten, insbesondere Zentraleinheiten, bieten die Möglichkeit, das Verhalten der Geräte über manuell vorgegebene Regelwerke zu steuern. Das sind die Policies. Umgekehrt, erlaubt auch der Xonbot deren Vorgabe - in

seiner Terminologie als Constraints. Das Regelwerk der Constraints im Xonbot kann prinzipiell beliebig komplex sein. Unter Benutzung der Zeit ist es auch möglich damit zeitliche Verläufe vorzugeben - zum Beispiel für Temperaturbereiche, welche im Raum einzuhalten sind.

Im Ergebnis schwankt die Realität zwischen zwei Extremen. Im ersten Extremfall sind die Aktionen der Geräteschaltungen durch die Policies völlig fest vorgegeben. Ein Beispiel dafür ist die Nutzung unserer aller ersten Helligkeitsregel als Policy: Wenn die Helligkeit kleiner 50% ist, die Lampe einschalten, sonst ausschalten. Da bleibt für den Xonbot nicht viel zu entscheiden. Der andere Extremfall gibt gar keine Policies vor, oder unrestriktive Policies wie

$$\text{if } (-\infty < \text{Helligkeit} < +\infty)$$
$$\text{then [mach' was du willst].}$$

Dann kann der Xonbot völlig frei agieren. In der Praxis wählt man meist einen Mittelweg.



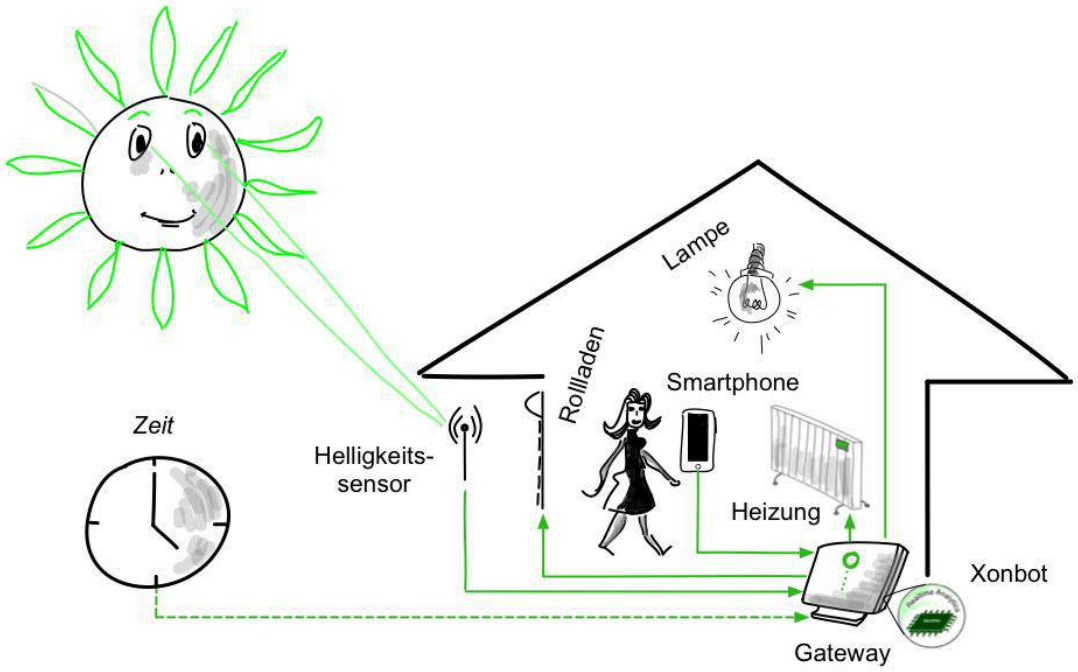
Den Grad der Autonomie der Maschine bestimmt der Mensch somit selbst: Je restriktiver die Policies, desto weniger Spielraum bleibt für die autonome Entscheidungsfindung des Xonbot, und umgekehrt.

Genug ist nicht genug

In der beschriebenen Weise können wir nun die verschiedensten Smart-Home-Geräte - Sensoren wie Aktoren - an den Xonbot anschließen und dieser wird lernen sie so miteinander zu verknüpfen, dass sie das Verhalten der Hausnutzer statistisch präzise beschreiben. Dabei können Geräte natürlich nicht nur ein- und ausgeschaltet werden, sondern auch gleitend.

Ein bekanntes Beispiel sind die Philips HUE Lampen, für die mittels des Xonbot auch sehr nützliche Applikationen angeboten werden. Hier kann nicht nur die Helligkeit der Lampe stufenlos reguliert werden, sondern auch Sättigung und Farbwert.

Im Ergebnis vereinfacht und verbessert der Xonbot deutlich die Smart-Home-Steuerung, welche bisher manuell geschah. Darüber hinaus können wir die Automatisierung noch für andere Zwecke wie Unterhaltung oder Anwesenheitssimulation zur Einbruchsprävention verwenden.



Nun kommen wir zu einem weiteren wichtigen Themenkomplex: dem Energiemanagement. So können wir auch in sinnvoller Weise Heizungen über den Xonbot steuern, da er befähigt ist gewünschte Temperaturverläufe zu erlernen.

Tafel der Besten

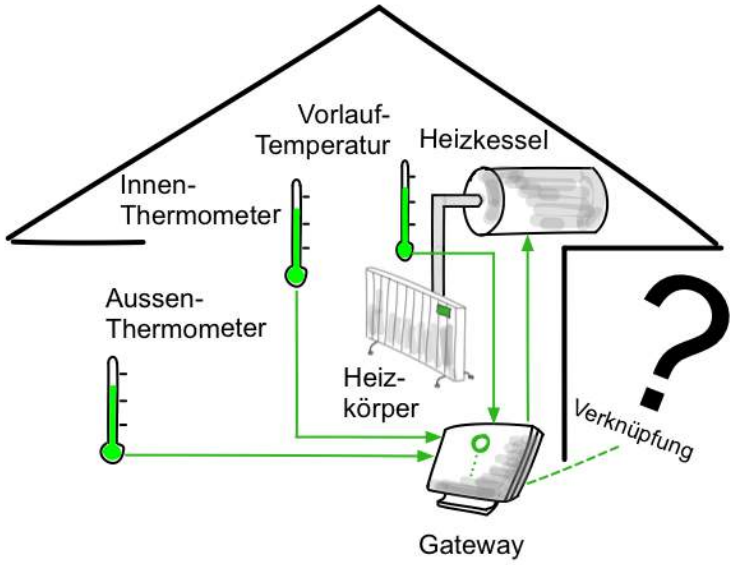
Ein Vorreiter dazu ist das Thermostat der amerikanischen Firma NEST. In einer ersten Phase manueller Nutzung beobachtet es den Nutzer und findet seine Heizgewohnheiten heraus. Danach kann es automatisch die Heizung entsprechend diesen Gewohnheiten vornehmen und dies mit externer Steuerung kombinieren. Findet es beispielsweise heraus, dass Steve am liebsten bei 21 Grad frühstückt, so regelt es rechtzeitig vorher die Temperatur auf 21 Grad. Über Sensoren erkennt es, wenn das Haus verlassen wurde und regelt automatisch die Temperatur herab. Auf diese Weise nimmt es Steve Arbeit ab und senkt zugleich die Heizkosten.

Die Fähigkeiten des Xonbot gehen dabei über jene derzeitiger selbstlernender Thermostate wie NEST hinaus: Da er nicht nur die Raumtemperatur kennt, sondern z.B. auch die Helligkeit, Nutzung von Haushaltsgeräten und Anwesenheit von Personen, kann er sehr viel genauer prognostizieren, welche Temperaturverläufe die Personen im Haushalt wünschen.

Der Xonbot kann jedoch noch deutlich mehr: Optimieren! Also automatisch lernen wie man eine Zielgröße minimiert (oder maximiert). Wir wollen beim Minimieren bleiben, denn Maximieren bedeutet nur das Negative der Zielgröße zu minimieren - ist also mathematisch das gleiche Problem. Damit nehmen wir wieder das Stichwort Energiemanagement auf.

Betrachten wir den einfachen Fall einer Außentemperatur-gesteuerten Warmwasserheizung. Hierbei wird das Wasser im Heizkessel auf die Vorlauftemperatur erwärmt, welches zu den Heizkörpern gepumpt wird. Diese geben durch Konvektion einen Teil der Wärmeenergie an

die Raumluft ab. Das abgekühlte Wasser fließt über die Rücklaufleitungen zurück zum Kessel. Weiterhin sollen die Außen- und Innenraumtemperatur über Thermometer erfasst werden. Das Ziel ist nun die möglichst energiesparende Steuerung des Heizkessels über dessen Vorlauftemperatur.



Um möglichst viel Energie zu sparen wird die Tatsache ausgenutzt, dass der Energieverbrauch für möglichst geringe Vorlauftemperaturen minimal ist.

Wir wollen das Problem zunächst mathematisch erfassen. Die Messdatenerfassung der Thermometer sowie die Neuregelung der Vorlauftemperatur geschehen täglich in N äquidistanten Zeitintervallen. Falls das Zeitintervall beispielsweise 10 Minuten beträgt ergibt sich N = 24 x 60 / 10 = 144 Zeitpunkte am Tag. Es sei eine konstante Innentemperatur vorgegeben - die Solltemperatur. Wir suchen nun den optimalen Verlauf der Vorlauftemperatur zu allen Zeitpunkten $i = 1, ..., N$.

Die Lösung dieses Problems beinhaltet dann im Kern zwei Aufgaben, die sich vereinfacht wie folgt ausdrücken lassen:

1. **Modellierung** des Zusammenhangs zwischen Vorlauf- und Innentemperatur,
2. **Minimierung** des Energieverbrauchs durch richtige Wahl der Vorlauftemperatur.

Beginnen wir mit der Modellierung. Die Innentemperatur hängt dabei mindestens von der Vorlauftemperatur und der Außentemperatur ab. Dieser Zusammenhang kann auch sehr komplex sein: So kann die Innentemperatur zum Zeitpunkt i auch von den vorherigen Verläufen ihrer selbst sowie der Vorlauf- und der Außentemperaturen abhängen. Es geht hier also primär um die Modellierung von Gebäudethermik und Außentemperaturprognose.

Die zweite Aufgabe der Minimierung wiederum führt auf ein Optimierungsproblem, in welchem das Modell aus 1. benutzt wird, um den optimalen Verlauf der Vorlauftemperatur über den Rest des Tages zu berechnen. Das geschieht so, dass zum einen die Summe der Vorlauftemperaturen zu allen betrachteten Zeitpunkten minimal ist und zum anderen die Abweichungen der Innentemperatur von der vorgegebenen Soll-Temperatur.

Einschub: Modellbasierte Lösung

Generell gilt: Sofern sich ein physisches Problem mit vertretbarem Aufwand physikalisch modellieren und lösen lässt (der Ansatz wird als "modellbasiert" bezeichnet), ist eine solche Lösung der statistischen vorzuziehen. Letztere entspricht dem Ansatz des Xonbot als selbstlernendem Datenanalysesystem.

Lässt sich das Optimierungsproblem also modellbasiert lösen? In der Tat kann die Temperatur in Abhängigkeit der Wärmequellen (also der Heizkörper) unter Nutzung der Geometrie des Hauses sowie seiner Materialeigenschaften physikalisch modelliert werden. Die zugrunde liegenden Differentialgleichungen sind seit vielen Jahren bekannt. So wird die Wärmeleitung durch die inhomogene instationäre Wärmeleitungsgleichung

$$\frac{\partial}{\partial\theta}t(\boldsymbol{x},\theta) - a\Delta t(\boldsymbol{x},\theta) = g(\boldsymbol{x},\theta) \qquad \text{(W)}$$

beschrieben, wobei $t(\mathbf{x},\theta)$ die Temperatur an der Stelle ***x*** des *d*-dimensionalen Raumes zum Zeitpunkt θ, Δ der Laplace-Operator bezüglich ***x*** und die Konstante *a* > 0 die Temperaturleitfähigkeit des Mediums ist. Die Wärmequellen werden durch $g(\mathbf{x},\theta)$ beschrieben und hängen wiederum von der Vorlauftemperatur ab.

Gleichung (W) sieht gar nicht so schlimm aus? Naja, für den Fall mit Null-Anfangsdaten $t_0(\mathbf{x}) = 0$ erhalten wir als Lösung:

$$t(\boldsymbol{x},\theta) = \int_0^\theta \int_{\mathbb{R}d} \frac{1}{(4\pi a\theta)^{\frac{d}{2}}} e^{-\frac{\|x-y\|^2}{4a\theta}} g(\boldsymbol{y},\eta)d\boldsymbol{y}d\eta.$$

Und das ist noch der einfachste Fall! Zusätzlich zur Wärmeleitung spielen für die Wärmeübertragung aber auch Konvektion und Wärmestrahlung eine Rolle. Auch diese werden über komplexe Differentialgleichungen beschrieben, darunter die Navier-Stokes-Gleichung, deren Lösung die Mathematiker seit über 150 Jahren beschäftigt. Das alles geschieht auch noch im Inneren des Hauses, dessen reale Geometrie die Randbedingungen sehr komplex macht. Kurz, es ist ein mathematisch hochkompliziertes Problem.

Und dass war nur die Modellierung! Nun muss mit dem Modell das Minimierungsproblem gelöst werden. Selbst wenn man bedenkt, dass solche Berechnungen von Ingenieuren durchaus erfolgreich ausgeführt werden (unter Nutzung von Vereinfachungen und numerischen Methoden), so bleibt der Aufwand dennoch enorm hoch. Er setzt eine genaue Modellierung des Gebäudes voraus und ist für unsere Aufgabenstellung bisher nur für Musterhäuser umgesetzt worden.

Das alles lässt die statistische Lösung über den Xonbot wieder in einem milderen Licht erscheinen.

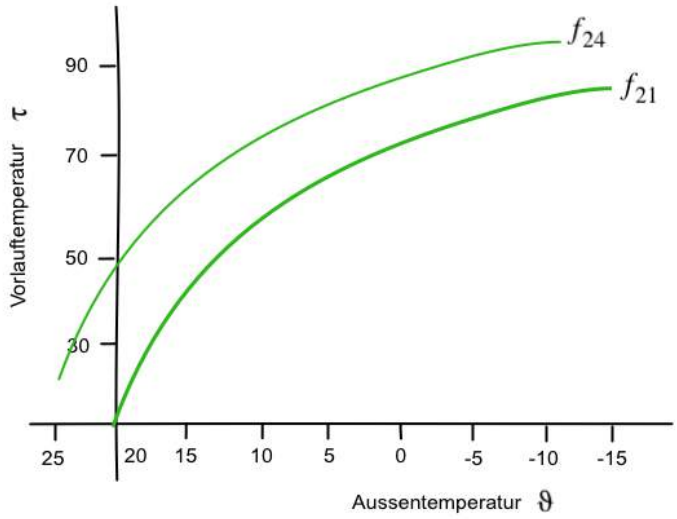
Probieren geht über studieren

Beginnen wir mit dem derzeit üblichen Verfahren der außentemperaturgeführten Vorlauftemperaturregelung. Hierbei wird für das konkrete Gebäude die Abhängigkeit der Innentemperatur *t* von der Vorlauftemperatur τ und der Außentemperatur ϑ experimentell ermittelt. Gesucht wird also der funktionelle Zusammenhang *h* mit $t = h(\tau, \vartheta)$; bezogen auf den Zeitschritt *i* also $t_i = h(\tau_i, \vartheta_i)$.

In der Praxis geschieht das über die sogenannte *Heizkurve*. Hierbei wird für eine konstante Innentemperatur t_C der Zusammenhang zwischen Außentemperatur und zugehöriger Vorlauftemperatur bestimmt. Mathematisch ausgedrückt folgt daraus, dass bei fixiertem t_C die Vorlauftemperatur als Funktion der Außentemperatur dargestellt werden kann:

$$\tau = f_{t_C}(\vartheta). \qquad \text{(H)}$$

Für verschiedene Innentemperaturen t_C lassen sich somit die Heizkurven grafisch abtragen. Im Bild sind exemplarisch für ein Haus zwei Heizkurven für jeweils °21 und °24 Innentemperatur abgetragen:



Natürlich ist es wünschenswert, dass eine Heizkurze möglichst niedrig ist, da dies in kleinen Vorlauftemperaturen resultiert, welche wenig Energie verbrauchen. Die Steilheit der Heizkurve bestimmt, wie stark eine Änderung der Außentemperatur eine Änderung der Vorlauftemperatur bewirkt. Eine Parallelverschiebung der Heizkurve nach oben entspricht - grob gesagt - einer Erhöhung der zugehörigen Innentemperatur t_C.

Wir wollen an dieser Stelle nicht weiter auf die Diskussion der Heizkurven eingehen. Gleichfalls nicht auf die verschiedenen Verfahren ihrer experimentellen Bestimmung. Zu erwähnen ist an dieser Stelle, dass mittlerweile etliche Anbieter von Heizreglern

automatisierte Verfahren zur Bestimmung der Heizkurven anbieten. Auf dieses Thema gehen wir im nächsten Einschub ein.

In der Anwendung der Heizungssteuerung wird die gewünschte Innentemperatur möglichst klein gewählt, also grob gesagt $t = t_{\min}$, da so die Vorlauftemperaturen am geringsten sind und somit auch der Energieverbrauch. Der Heizregler passt dann die Vorlauftemperatur entlang der Heizkurve (H) automatisch an die Außentemperatur an.

Bleibt eine Frage. Löst die außentemperaturgeführten Vorlauftemperaturregelung tatsächlich unser Minimierungsproblem? Es lässt sich zeigen, dass das in der Tat der Fall ist.

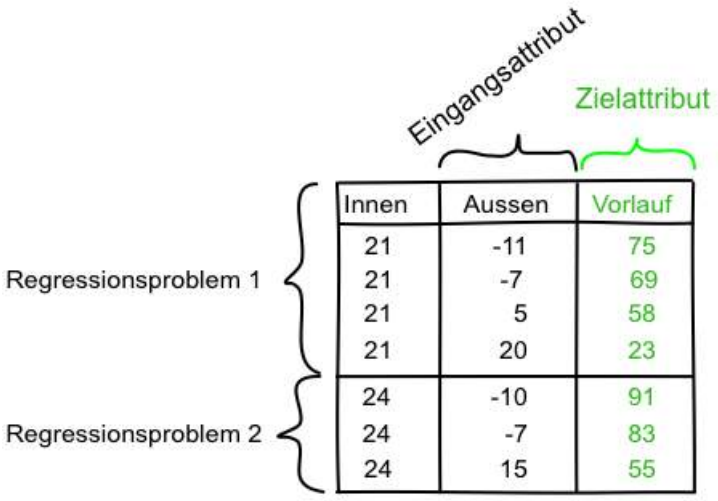
Einschub: Regression

Bisher haben wir nur nebulös über die experimentelle Bestimmung der Heizkurven gesprochen. Wir wollen das jetzt konkretisieren.

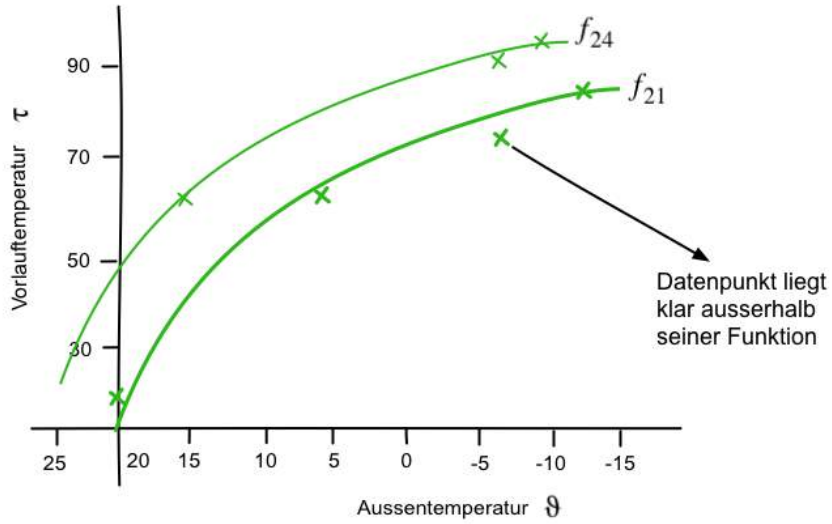
Das geschieht im Kern über das Verfahren der *Regression*. Diese spielt für den Xonbot eine zentrale Rolle. Aufgabe der Regression ist es anhand von einzelnen Datenpunkten einen funktionellen Zusammenhang zwischen mehreren Variablen - den *Eingangsvariablen* - und einer weiteren Variable - der *Zielvariable* - zu finden. Die Datenpunkte werden dabei auch als *Datenvektoren* und die Variablen als *Attribute* bezeichnet.

Es ist intuitiv klar, dass die gesuchte Funktion möglichst die Datenvektoren beschreiben sollte. Das "möglichst" werden wir noch konkretisieren.

Der anschaulichste Fall ist die eindimensionale Regression, bei der nur eine Eingangsvariable existiert und wir eine Funktion suchen, welche die Eingangsvariable auf die Zielvariable abbildet. Genau das wird für die Bestimmung der Heizkurve benötigt. Zur Illustration betrachten wir unsere vorherigen Heizkurven. Angenommen es wurden folgende Datenpunkte gemessen:



Dabei haben wir die Datenpunkte bereits nach gleichen Innentemperaturen sortiert. Im Ergebnis erhalten wir zwei Regressionsprobleme: Aus den vier Datenpunkten für °21 konstruieren wir die Regressionsfunktion $f_{21}(\vartheta)$ und aus den drei Datenpunkten für °24 konstruieren wir $f_{24}(\vartheta)$. Im Ergebnis erhalten wir unsere beiden Heizkurven:



Hier sind die Datenpunkte als kleine Kreuze eingezeichnet. Wir sehen, dass es im allgemeinen nicht möglich ist, die Funktion eines Regressionsproblems so zu konstruieren, dass sie exakt durch jeden Datenpunkt geht.

Wie finden wir die Regressionsfunktion? Dazu formalisieren wir das Regressionsproblem. Unsere Datenpunkte lassen sich als Datenmatrix *S* ausdrücken:

$$S = \{(\boldsymbol{x_i}, y_i) \in \mathbb{R}^d \times \mathbb{R}\}_{i=1}^{M}. \qquad \text{(S)}$$

Dabei entspricht jede Zeile $(\mathbf{x}_i, y_i)$ dem Vektor des *i*-ten Datenpunkts und jede Spalte einem Attribut, wobei $\mathbf{x}_i$ der Vektor der Eingangsattribute ist und y_i der Wert des Zielattributs. Die Anzahl der Datenpunkte sei dabei *M* und die Anzahl der Eingangsattribute *d*. In unserem Beispiel betrug *M* für das erste Regressionsproblem 4 und für das zweite 3. Die Anzahl der Eingangsattribute *d* war in beiden Fällen 1.

Gesucht ist nun eine Regressionsfunktion

$$f(\boldsymbol{x}) = y. \qquad \text{(R)}$$

Um diese zu konstruieren, gehen wir davon aus, dass sie die Datenvektoren aus (S) möglichst gut beschreibt; der Mathematiker spricht hier von *Approximation*. Damit kommen wir zur Beschreibung des "möglichst" zurück. Ein üblicher Ansatz ist hierzu die Minimierung des quadratischen Fehlers:

$$\min_{f \in V} \sum_{i=1}^{M} (f(x_i) - y_i)^2. \qquad (Q)$$

Wir suchen also diejenige Funktion *f* eines vorgegebenen Funktionsraums *V*, welche die Summe der quadratischen Fehler über allen Datenvektoren minimiert. Sofern wir die Regressionsfunktion im Raum der linearen Funktionen suchen erhalten wir die klassische Formulierung der *linearen Regression*. Suchen wir im Funktionsraum beliebiger Polynome, erhalten wir die *polynomiale Regression*. Usw.

Es existieren zahlreiche weitere Regressionsverfahren wie Regressionsbäume, Regressions-Splines, Neuronale Netze, Support-Vektor-Regression, Dünngitterregression, etc. Diese lösen oft komplexere Formulierungen, insbesondere in Form von Regularisierungsnetzwerken, welche die Formulierung (Q) verallgemeinern. Wir wollen das Thema Regression an dieser Stelle nicht weiter vertiefen und verweisen stattdessen auf die umfassende Literatur hierzu.

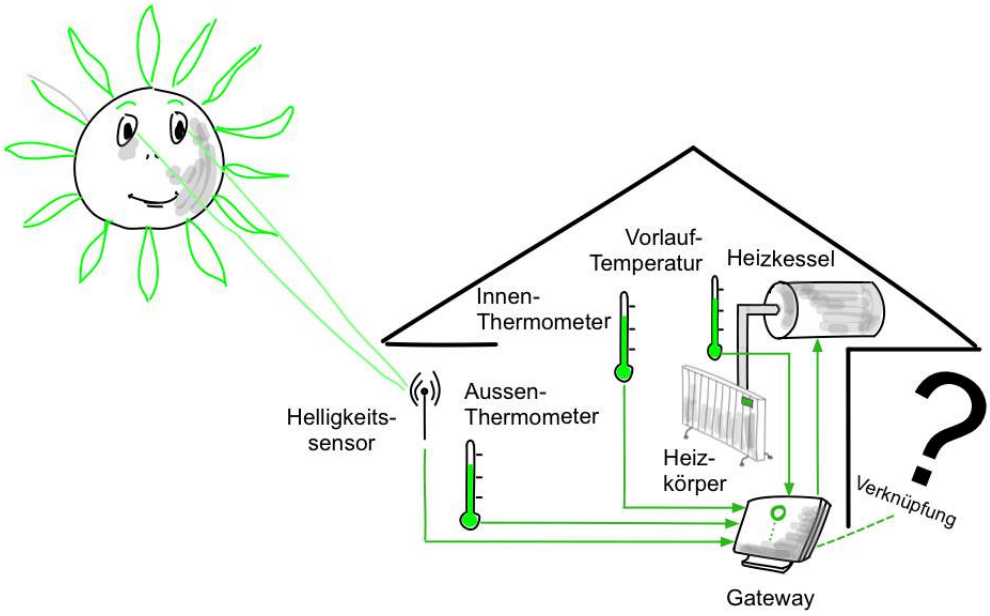
Abschließend weisen wir noch darauf hin, dass die Regressionsverfahren im Xonbot Probleme wie (Q) durch Aktualisierung von *f* für jeden Datenvektor aus (S) lösen können, also inkrementell. Daraus resultieren wieder die am Beispiel der Mittelwertberechnung beschriebenen Vorteile: Anpassung von *f* in Echtzeit und (S) muss nicht gespeichert werden.

Die vorausschauende Heizung

Die beschriebene außentemperaturgeführten Vorlauftemperaturregelung löst wie dargelegt unsere Minimierungsaufgabe. Auch erweist sie sich in der Praxis als sehr erfolgreich, warum sie sich auch durchgesetzt hat.

Dennoch weist sie Schwachpunkte auf. Ihr zentraler Schwachpunkt ist die vereinfachte Modellierung der Innentemperatur, welche nur in Abhängigkeit der Außentemperatur modelliert wird. Zugleich geschieht diese Modellierung in stark vereinfachter Weise. So sollten weitere Einflüsse wie Sonneneinstrahlung, Verhalten der Bewohner und Gebäudeeigenschaften berücksichtigt werden.

Wir wollen daher exemplarisch noch die Sonnenstrahlung in unser Modell aufnehmen. Es zeigt sich, dass die Lösung des Minimierungsproblems nun nicht mehr einfach dadurch möglich ist, dass als Innentemperaturen immer die Soll-Temperatur gewählt wird. Um das zu sehen betrachten wir die Sonneneinstrahlung am Nachmittag. So kann es sinnvoll sein, in Zeiten intensiver Sonnenstrahlung die Innentemperatur anzuheben, um das Haus mit weniger Energie aufzuheizen. Diese erlaubt uns dann bei abnehmender Sonnenstrahlung überproportional weniger Energie durch niedrigere Vorlauftemperaturen zu verbrauchen. Kurz gesagt: Eine zwischenzeitliche Erhöhung der Vorlauftemperaturen und damit der Innentemperaturen über das nötige Mindestmaß hinaus führt letztlich zu einer niedrigeren Summe aller Vorlauftemperaturen über den gesamten Tag.



Es gibt nun zwei Ansätze: Über Regressionsverfahren kann aus den Daten der Temperaturen und Sonneneinstrahrung zu allen Zeitpunkten ein Modell gelernt werden, welches aus deren bisherigen Verläufen in jedem Schritt die Innentemperatur vorhersagt. Mit diesem Modell kann dann das Minimierungsproblem gelöst werden, welches den optimalen Verlauf der Vorlauftemperaturen über den gesamten Tag berechnet.

Einen weitaus flexibleren Ansatz benutzt der Xonbot: Reinforcement Learning.

Einschub: Optimierung über Reinforcement Learning

Fassen wir mal unsere bisherigen Anforderungen an den Xonbot zusammen. Dieser soll

1. in Echtzeit ein Analysemodell seiner Interaktion mit der Umwelt lernen,
2. in Echtzeit daraus Entscheidungen zur Minimierung einer Zielgröße ableiten,
3. und das so, dass die Minimierung über die Summe aller nächsten Schritte geschieht.

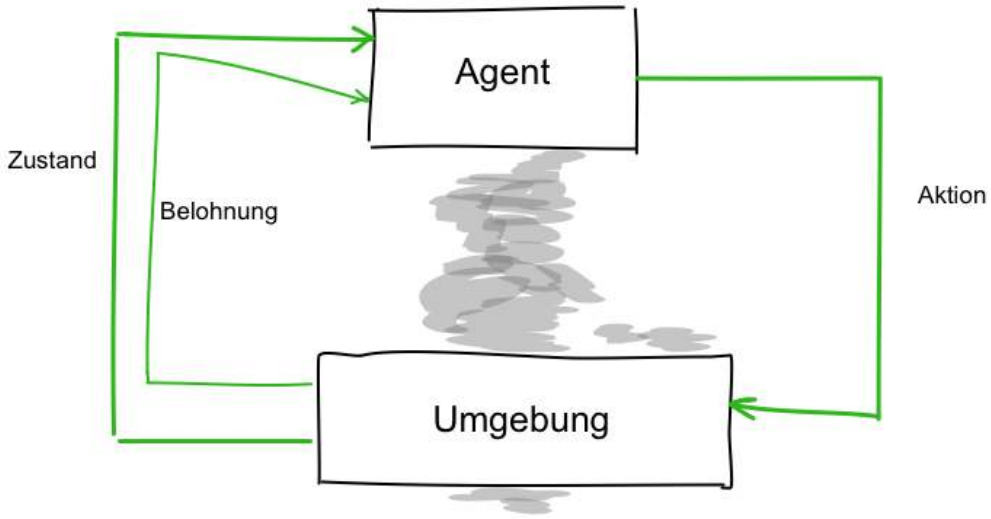
Ein solcher Ansatz kann über *Reinforcement Learning* (RL) geschehen. Reinforcement Learning (bestärkendes Lernen) beschäftigt sich damit, Systeme zu entwickeln (sogenannte Agenten), die über ein Wechselspiel mit der Umgebung schrittweise lernen, eine skalare Zielgröße zu maximieren. Beispiele sind Steuerungen autonomer Roboter, Simulationen in der Wirtschaft und selbstlernende Programme für Spiele wie Schach. RL hat seinen Ursprung in der mathematischen Disziplin der *Dynamischen Programmierung* (DP), welche in der *Regelungstheorie* angewendet wird. Im Gegensatz zu DP liegt der Schwerpunkt des RL jedoch auf dem Gebiet des adaptiven Lernens.

Obwohl über die Jahre große Verbesserungen im Bereich des RL erzielt wurden, hielt sich die Zahl seiner praktischen Applikationen bisher stark in Grenzen. Das lag primär an der hohen Komplexität seiner mathematischen Verfahren. Dennoch setzt sich RL langsam durch.

Tafel der Besten

Ein bekanntes Beispiel ist das Computerprogramm *AlphaGo*, das ausschließlich das Brettspiel Go spielt. Es wurde von Google DeepMind entwickelt. Im Januar 2016 wurde bekannt, dass AlphaGo bereits im Oktober 2015 den mehrfachen Europameister Fan Hui besiegt hatte. Damit ist es das erste Programm, das unter Turnierbedingungen ohne Vorgabe (Handicap) auf einem 19×19-Brett einen professionellen Go-Spieler schlagen konnte. Im März 2016 schlug AlphaGo den Südkoreaner Lee Sedol, der als einer der weltbesten Profispieler angesehen wird (AlphaGo gegen Lee Sedol). AlphaGo kombiniert Reinforcement Learning und Traversierung.

Kommen wir zur mathematischen Modellierung des RL. Dazu betrachten wir einen Agenten, der mit seiner Umwelt kommuniziert. So besitzt die Umwelt eine Menge von *Zuständen*. Der Agent kann in jedem Zustand *s* eine *Aktion a* aus einer Aktionsmenge wählen und gelangt so in einen Folgezustand *s'* und erhält dabei eine *Belohnung r*.



Ziel des Agenten ist es den *erwarteten Gewinn*

$$R_\theta = \sum_{k=0}^{T} \gamma^k r_{\theta+k+1} \qquad \text{(B)}$$

im aktuellen Zeitschritt θ zu maximieren. Der erwartete Gewinn ist also so etwas wie die erwartete Gesamtbelohnung. Dabei nennt man γ den *Diskontierungsfaktor*, der zukünftige Belohnungen gewichtet. Bei *episodischen Problemen*, d. h. die Umwelt geht nach einer endlichen Anzahl von Schritten in einen Endzustand über (wie z.B. eine Schachpartie), eignet sich der Diskontierungsfaktor $\gamma = 1$. In diesem Fall wird jede Belohnung $r_{\theta+k+1}$ gleich

gewertet. Bei *kontinuierlichen Problemen* ($T = \infty$) muss man ein $\gamma < 1$ wählen, damit die unendliche Reihe R_θ konvergiert. Für $\gamma = 0$ zählt nur die aktuelle Belohnung $r_{\theta+k+1}$; alle zukünftigen Belohnungen werden ignoriert. Geht γ gegen 1, wird der Agent weitsichtiger.

Die Wahl der Aktion *a* in jedem Zustand *s* durch den Agent jeisst *Policy* (Strategie) π. Das Ziel des Agenten besteht darin, eine Policy zu finden, welche den erwarteten Gewinn maximiert. Diese wird als *optimale Policy* $\pi*$ bezeichnet.

Im allgemeinen wird ein RL-Problem als *Markow-Entscheidungsproblem* formuliert. Das bedeutet, dass die Wahl der richtigen Aktion nur vom aktuellen Zustand *s* abhängt. Ein gutes Beispiel für ein Problem, welches der Markow-Eigenschaft genügt, ist wieder das Schachspiel. Um in jeder Stellung den besten Zug auszuführen, ist es mathematisch gesehen völlig unerheblich, wie die Stellung aufs Brett gekommen ist (wenngleich in der Spielpraxis meist hilfreich). Hingegen ist es wichtig, alle möglichen Folgetransaktionen für jeden Zug durchzurechnen (was natürlich in der Praxis meist nur für eine gewisse Spieltiefe möglich ist), um den optimalen Zug zu finden.

Die Markow-Annahme erlaubt uns die Komplexität der Modellierung drastisch zu senken. Zugleich muss natürlich in jedem Fall geprüft werden, ob die Markow-Eigenschaft hinreichend erfüllt ist. Damit können wir nun die Policy als $\pi(s)$ aufschreiben. Weiterhin sind nichtdeterministische Policies (gemischte Strategien) $\pi(s,a)$ möglich, sodass eine Aktion mit einer bestimmten Wahrscheinlichkeit ausgewählt wird. Im Allgemeinen wird eine Strategie demnach als bedingte Wahrscheinlichkeitsverteilung definiert: $\pi(s,a) = p(a \mid s)$.

Zur Umsetzung der Policy wird eine Zustandswertfunktion $f(s)$ benötigt, die jedem Zustand den erwarteten Gewinn zuweist. Sofern die Übergangswahrscheinlichkeiten nicht explizit bekannt sind, wird sogar eine Aktionswertfunktion $f(s,a)$ benötigt, welche jedem Zustand *s* und jeder darin zulässige Aktion *a* den erwarteten Gewinn zuweist. Die Funktion *f* wiederum löst ein Regressionsproblem, wodurch die zentrale Bedeutung der Regression für RL ersichtlich wird. Die Umsetzung der Policy $\pi(s)$ läuft also im Kern auf die Wahl der Aktion mit dem höchsten Aktionswert hinaus:

$$\pi(s) = \operatorname{argmax}_{a \in A(s)} f(s,a), \qquad \text{(P)}$$

wobei *A*(*s*) die Menge aller im Zustand *s* zulässigen Aktionen beschreibt. Eine solche Policy wird auch als *greedy* (gierig) bezeichnet. Ein solches Vorgehen heißt *Exploitation*, also Ausnutzung. Die effiziente Umsetzung von (P) gestaltet sich für eine kleine Aktionsmenge *A*(*s*) einfach; für komplexe Aktionsräume kann sie jedoch - in Abhängigkeit des gewählten Funktionsraums - sehr anspruchsvoll werden. Erst recht, wenn noch Constraints vorgegeben werden. Die effiziente Lösung von (P) unter Nebenbedingungen stellt eine der größten Stärken des Xonbot dar.

Darüber hinaus können Policies auch so gestaltet werden, dass sie (in geringerem Maße) neue Aktionen austesten und so die Wissensbasis erweitern. Dann nennt man das Vorgehen *Exploration*, also Erforschung. Dieses Austesten kann zufällig, aber auch systematisch erfolgen. Letzteres ist dann der *statistischen Versuchsplanung* vergleichbar. Hierbei soll mit

möglichst wenigen Experimenten, sprich explorativen Aktionen, ein Maximum an neuem Wissen über die Umwelt ermittelt werden. Das richtige Zusammenspiel von Exploitation und Exploration ist ein wichtiger Bestandteil des RL. Damit wollen wir den Überblick über Reinforcement Learning beenden und verweisen auf die zugehörige Literatur.

Betrachten wir nun die Modellierung unseres Minimierungsproblems zur Bestimmung der Vorlauftemperaturen als RL-Problem. Die Epochen sind hier immer die einzelnen Tage, ihre Länge *T* = *N*. In jedem Zeitschritt gelangt das System in einen neuen Zustand, es gilt also $\theta = i$. Ein Zustand s_i wird dabei durch die aktuellen Temperaturen und Sonneneinstrahlung beschrieben, einschließlich ihrer bisherigen Verläufe bzw. deren Aggregaten. Eine Aktion a_i entspricht der neuen Vorlauftemperatur. Die Belohnung r_i in jedem Schritt *i* ergibt sich als negative gewichtete Summe der Abweichung der Ist- von der Solltemperatur und der Vorlauftemperatur. Das negative Vorzeichen wird benötigt, weil RL den erwarteten Gewinn maximiert, wir aber minimieren. Es gilt $\gamma = 1$. Somit können wir das komplette Minimierungsproblem als RL-Problem lösen. Genau das übernimmt der Xonbot.

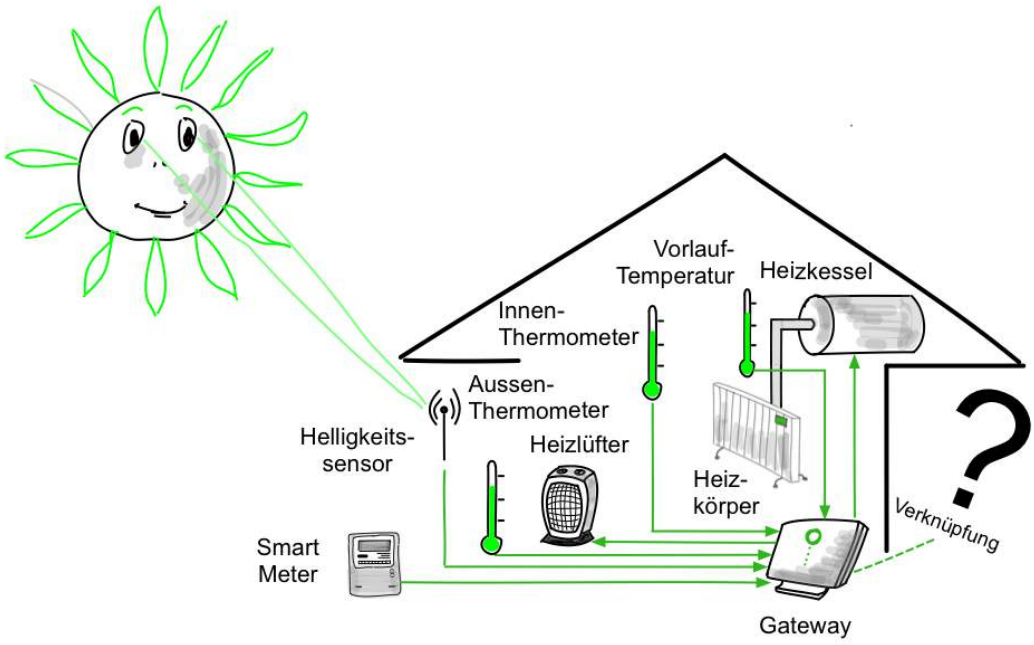
Und dank der Flexibilität des Ansatzes kann der Xonbot noch mehr. Die daraus resultierenden Vorteile beschreiben wir als nächstes.

Was geht noch?

Wie geht der Xonbot also vor, um die möglichst energiesparende Aussteuerung der Heizung für eine vorgegebene Innenraumtemperatur zu erlernen? Er registriert in jedem Zeitschritt die Temperaturen und Sonneneinstrahlung, aktualisiert seine Aktionswertfunktion $f(s, a)$ (ein komplexer Prozess, der eine inkrementelle Regressionsberechnung beinhaltet) und setzt damit die Policy gemäß (P) um, welche als Aktion die nächste Vorlauftemperatur zurückliefert. Das wiederholt sich für jeden weiteren Zeitschritt bis zum Ende des Tages. Dann wird eine neue Episode angelegt und alles beginnt von vorn. Im Ergebnis lernt der Xonbot immer besser den Energieverbrauch zu minimieren.

Nun kommt die Stärke des Reinforcement Learning ins Spiel: seine gewaltige Flexibilität. Angenommen wir nutzen im Haus noch einen Heizlüfter, der im einfachsten Fall über einen Smart Plug ein- und ausgeschaltet werden kann. Weiterhin sei ein Smart Meter verfügbar, der dessen Stromverbrauch messen kann.

Dann können wir den bestehenden Ansatz leicht erweitern. Zum einen nehmen wir nun in jedem Zeitschritt den Stromverbrauch einschließlich Verlauf bzw. Aggregaten mit in den aktuellen Zustand s_i auf. Außerdem erweitern wir den Aktionsraum um die Schaltung des Heizlüfters, so nun jede Aktion a_i aus dem Tupel der neuen Vorlauftemperatur und Ein- oder Ausschaltung des Heizlüfters besteht.



Außerdem müssen wir die "Belohnung" in jedem Schritt anpassen. Da hier die Belohnung ja eigentlich eine Bestrafung ist, bedeutet das die zusätzlichen Aufnahme des Stromverbrauchs. Somit besteht nun die Belohnung r_i aus der negativen gewichteten Summe der Innentemperaturabweichung, der Vorlauftemperatur und des Stromverbrauchs.

Das Ganze lässt sich nun im Prinzip beliebig erweitern, z.B. um folgende Funktionalitäten:

Erweiterung	Beschreibung
Raumtemperaturen regeln	Thermometer in allen Räumen, jeder mit eigener Solltemperatur, dazu alle Heizkörperventile steuern
Weitere Heizgeräte einbeziehen	Konvektoren, Ölradiatoren, Heizstrahler, etc. steuern
Solaranlagen einbeziehen	Energieerzeugung durch Solaranlagen messen und Einspeisung steuern
Soll-Temperaturverläufe vorgeben	Vorgabe zeitlicher Temperaturverläufe als Sollgrösse, zusätzlich Integration von Komforteinstellungen
Soll-Temperaturen mit automatischer Erkennung	Wie Soll-Temperaturverläufe, aber diese werden automatisch aus Benutzerverhalten gelernt
Bewegungsmelder einbeziehen	Einbeziehung zur Verbesserung der Prognose des Benutzerverhaltens
Warmwasserspeicher einbeziehem	Messung und Steuerung

Theoretisch lässt sich das beliebig fortsetzen. Alle Arten von digitalen Messgeräten und Sensoren können mit in die Auswertung aufgenommen werden. Und umgekehrt können alle nur denkbaren energierelevanten Geräte ausgesteuert werden. Im Ergebnis ließe sich ein

umfassendes selbstlernendes Energiesparsystem konfigurieren, welches alle nur denkbaren Aspekte berücksichtigt.

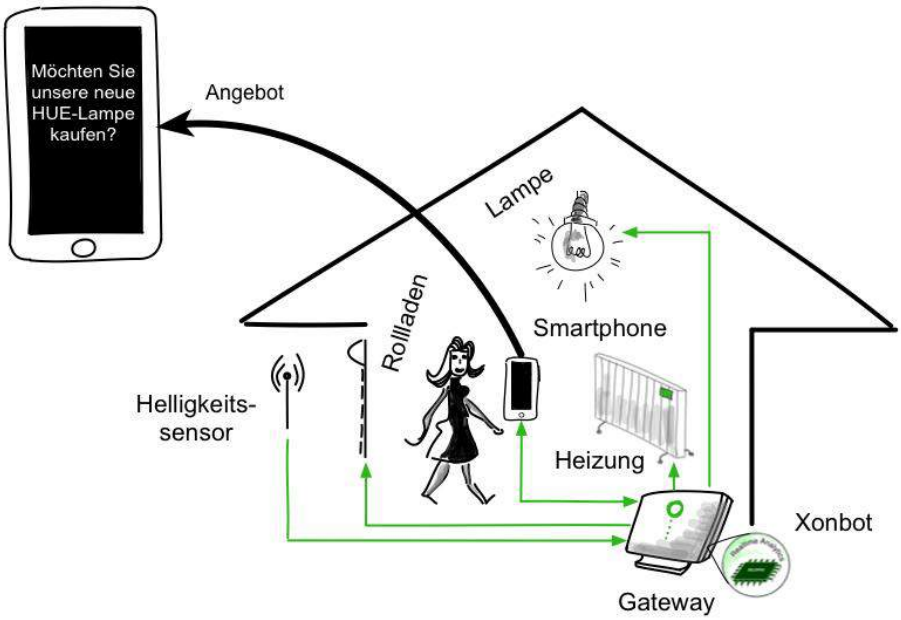
Theoretisch, wohlgemerkt. In der Praxis hatten wir ja gesehen, dass einiges zu tun ist: Die Messdaten müssen in die Zustände aufgenommen werden, samt Verläufen und/oder Aggregaten. Die Aggregate (mittlere bisherige Temperatur, maximale bisherige Sonneneinstrahlung, etc.) verbessern das Lernverhalten des Xonbot und ihre Festlegung erfordert intellektuelle Arbeit. Die Belohnung muss sinnvoll erweitert werden. Tests sind notwendig, etc. Zum anderen kann eine zu große Komplexität dazu führen, dass das System instabil arbeitet. Dieses Thema werden wir noch separat diskutieren.

Es ist also alles nicht so einfach. Gleichwohl ermöglicht die Nutzung des RL über den Xonbot eine viel schnellere Lösung verschiedenster Aufgabenstellungen als bisher.

Was Transaktionen alles noch sagen

Bisher hatten wir die Messdaten und das Schaltverhalten nur genutzt, um letzteres zu automatisieren bzw. zu optimieren. Da geht aber noch mehr! Zum Beispiel im Bereich Marketing. Aus dem Schaltverhalten lassen sich Rückschlüsse auf die Vorlieben der Nutzer herstellen. Dabei werden aus dem Nutzerverhalten typische Profile in Abhängigkeit der Vorlieben gebildet und dann zur Prognose der letzteren gebildet. Der Ansatz dazu ist die *Klassifikation*.

So kann ein Provider den Xonbot benutzen, um den Haushalten interessante Produkte zum Kauf anzubieten, z.B. die neueste HUE-Lampenkollektion oder deren Bewegungsmelder. Dabei werden nur jene Haushalte angesprochen, die potenziell ein hohes Interesse an dem Angebot haben und die anderen nicht belästigt.



Basis ist hierbei die Information, welcher Haushalt in der Vergangenheit bereits die Lampen erworben hat. Daraus werden mittels Klassifikation typische Käuferprofile gebildet. Danach

werden diejenigen Haushalte gezielt angesprochen, die eine hohe Kaufwahrscheinlichkeit haben.

Das Ganze funktioniert natürlich auch für andere Zielgrößen wie Produktkategorien, Reisen, etc. Es können auch Services angeboten werden, oder einfach nur Tipps. Der Xonbot lässt sich so schrittweise zum Berater erweitern. Mehr noch. Es lässt sich auch prognostizieren, ob ein Haushalt potenziell den Provider wechseln möchte. Die gefährdeten Kunden können dann zielgerichtet angesprochen werden, um den Wechsel zu vermeiden. Die Möglichkeiten der Klassifikation sind gewaltig.

Einschub: So funktioniert Klassifikation

Die Klassifikation kann formal als Spezialfall der Regression aufgefasst werden, die wir bereits vorgestellt hatten. Dabei nimmt das Zielattribut nur eine beschränkte Anzahl von Werten an, zumeist zwei. Regressions- und Klassifikationsverfahren werden oft auch als *Scoringverfahren* bezeichnet.

Wir betrachten daher wieder eine Anzahl von *M* Datenvektoren, welche über einen Satz von *d* Attributen beschrieben werden. Die Datenvektoren sind bei uns die Transaktionen, welche über Attribute wie Personenzahl im Haushalt, Anzahl der Lampen, Energieverbrauch, etc. beschrieben werden. Das Zielattribut beinhaltet die Vorlieben. Im Ergebnis erhalten wir wieder eine Datenmatrix gemäß (S).

Illustrieren wir das an unserem einfachen Beispiel: Wir wollen prognostizieren, ob sich Familie Meyer für das Angebot einer neuen Lampe interessiert. Dazu betrachten wir mehrere Haushalte und beschreiben sie durch zwei Attribute: die Anzahl der Lampen im Haushalt und die durchschnittliche Anzahl der täglichen Lampenschaltungen. Es soll weiterhin für jeden Haushalt bekannt sein, ob er schon einmal eine neue Lampe bestellte oder nicht. Ersteres soll mit +1 kodiert werden, zweites mit -1. Die Ausprägungen werden auch als *Klassen* bezeichnet. Im Ergebnis erhalten wir folgende Datenmatrix *S*:

	Eingangsattribute		Zielattribut
Haushalt	Lampen	Schaltung/Tag	Neu
Schmidt	15	20	-1
Popow	40	94	1
Federstein	22	48	1

So enthält der Haushalt von Herrn Schmidt 15 Lampen und er führt durchschnittlich 20 mal am Tag Lampenschaltungen aus. Eine neue Lampe hat er noch nie bestellt. Der Haushalt der Popows besitzt 40 Lampen; sie schalten durchschnittliche 94 mal täglich und die Familie hat bereits eine neue Lampe bestellt. Das können wir für beliebig viele Haushalte fortsetzen.

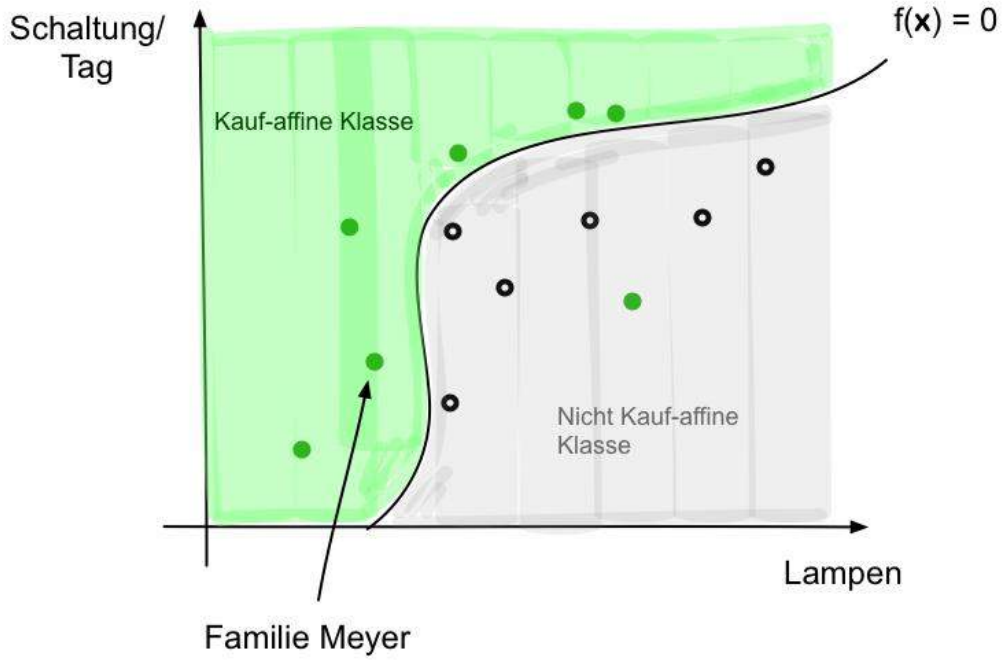
Wir wollen dabei das Attribute "Haushalt" ignorieren, da es zu spezifisch ist und auch nicht im Sinne des Datenschutzes, und das Zielattribut in Abhängigkeit der Eingangsattribute

"Lampen" und "Schaltung/Tag" grafisch abtragen. Dabei sollen grüne Punkte der ersten Klasse entsprechen (neue Lampe) und graue Punkte der zweiten Klasse (keine neue Lampe).

Die Aufgabe der Klassifikation besteht nun darin eine Funktion $f(\mathbf{x})$ zu finden, den *Klassifikator*, die in Abhängigkeit des Vektors der Eingangsattribute ***x*** möglichst gut den Wert des zugehörigen Zielattributs *y* vorhersagen kann. Damit suchen wir wie in (R) die Funktion

$$f(\boldsymbol{x}) = y.$$

Für unser Beispiel bedeutet das also, dass für den Haushalt Schmidt der Funktionswert über dem Eingangsvektor $\mathbf{x}_1 = (15, 20)$ möglichst nahe bei -1 sein soll, also $f(\mathbf{x}_1) \approx -1$. Für Familie Popow, also $\mathbf{x}_2 = (40, 94)$ möglichst nahe bei +1: $f(\mathbf{x}_2) \approx +1$ usw.



Unser gesuchter Klassifikator $f(\mathbf{x})$ ist somit eine Funktion über dem zweidimensionalen Raum. Alle Vektoren $\mathbf{x}$, für die gilt $f(\mathbf{x}) > 0$, werden somit der ersten Klasse zugerechnet und alle Vektoren mit $f(\mathbf{x}) \leq 0$ der zweiten Klasse. Offensichtlich können wir damit allen Vektoren (also Punkten) des zweidimensionalen Raums $\mathbf{x} \in \Re^2$ eindeutig eine Klasse zuordnen. Diese sei für die erste Klasse grün eingefärbt und für die zweite Klasse grau. Die Trennlinie zwischen den beiden Bereichen besteht offensichtlich aus allen Punkten $\mathbf{x}_S \in \Re^2$, für die gilt:

$$f(\boldsymbol{x_S}) = 0.$$

Nun illustriert unsere Abbildung den Begriff des „Lernens“: Indem wir auf der Menge *S* unseren Klassifikator *f* konstruieren, verallgemeinern wir das Wissen aus deren Vektoren. Nämlich so, dass wir nun jeden neuen zweidimensionalen Datensatz $\mathbf{x}$ einer der beiden Klassen zuordnen; je nachdem, ob er in den grünen oder grauen Bereich fällt.

Um nun zu prognostizieren, ob ein Haushalt affin für das Angebot einer neuen Lampe ist, setzen wir seinen Vektor $\mathbf{x}$ in die Klassifikatorfunktion *f* ein. Ist diese kleiner 0, so ist er wahrscheinlich nicht affin, ist er größer 0 so ist er umgekehrt affin. Je kleiner der Wert, desto geringer die Kaufaffinität und je größer desto höher. Angenommen Familie Meyer hat 35 Lampen und schaltet durchschnittlich 82 mal am Tag:

	Eingangsattribute		Zielattribut
Haushalt	Lampen	Schaltung/Tag	Neu
Meyer	35	82	?

Dann erhalten wir als Eingangsvektor $\mathbf{x} = (35, 82)$. Sei nun der Funktionswert $f(\mathbf{x}) = 1{,}6$ und somit deutlich > 0. Dann ist Familie Meyer wahrscheinlich kaufaffin und ihr sollte ein Angebot über Lampen unterbreitet werden.

Zur Klassifikation können prinzipiell Regressionsverfahren eingesetzt werde. Darüber hinaus gibt es aber etliche spezielle Klassifikationsverfahren wie Nächste Nachbarn, Entscheidungsbäume, Bayessche und Neuronale Netze, Support Vektor Maschinen, Dünne Gitter, etc. Wir wollen hier nicht weiter darauf eingehen, da das den Rahmen des Dokumentes sprengen würde und für das prinzipielle Verständnis nicht entscheidend ist.

Wichtig sind in dem Zusammenhang zwei Aspekte: Erstens nutzt der Xonbot inkrementelle Scoringverfahren, die also adaptiv lernen können ohne die Datenmatrix *S* abzuspeichern. Zweitens nutzt der Xonbot auch dafür das bereits beschriebene Framework des Reinforcement Learning. In der Tat stellt dessen Herzstück - die Aktionswertfunktion $f(s, a)$ - eine Regressionsfunktion dar.

Damit lässt sich die Aufgabe der Marketingautomatisierung sogar noch erweitern: Es geht nicht nur darum möglichst viele Nutzer zum Kauf eines Produkts anzuregen. Vielmehr erlaubt RL ja die Maximierung einer Zielgröße über mehrere Schritte hinweg! So kann der Umsatz oder Ertrag als Zielgröße benutzt werden, um über passende Angebote an die Haushalte eine Maximierung der Zielgröße zu erreichen. Der Xonbot ist für diese Aufgabe bestens geeignet, da er aus dem Bereich des Handels stammt: Dort löst er die Aufgabe den Nutzern im Handel das richtige Produkt zum richtigen Preis zur richtigen Zeit zu unterbreiten, um den Kundenwert zu maximieren.

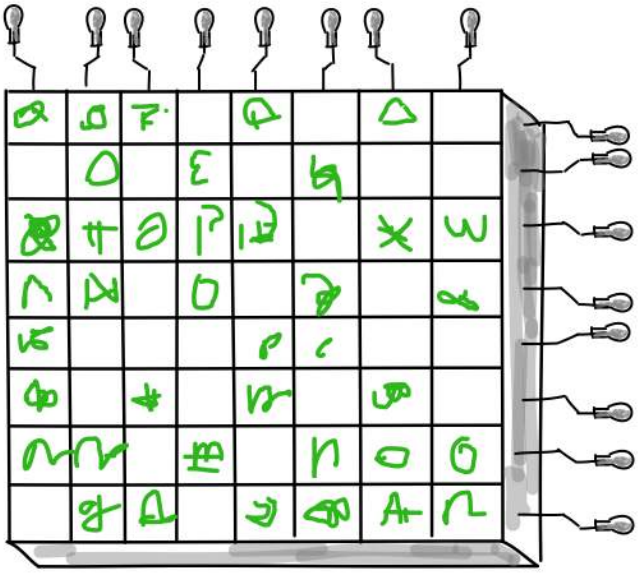
Die Nutzung des Reinforcement Learning potenziert somit nochmal die Anwendungsmöglichkeiten der Klassifikation! Zugleich macht es seine Nutzung nachhaltiger, da es "langfristig" agiert. Das wirft die Frage auf: Wo liegt eigentlich die Grenze der Anwendbarkeit des Xonbot?

In der Akademie von Lagado

Fassen wir nochmals zusammen. Mittels seines Reinforcement-Learning-Ansatzes ist der Xonbot in der Lage die Maximierung prinzipiell beliebiger Zielgrößen aus der Interaktion mit Nutzern und Umwelt in Echtzeit zu erlernen. Was liegt da näher als den Xonbot sämtliche Geräte und Marketingaktivitäten steuern zu lassen und als Zielgröße z.B. die Nutzerzufriedenheit zu definieren oder den insgesamt mit den Haushalten erzielten Umsatz?

Da das System ja selbstlernend ist, wird es schon alles allein rausfinden. Und entspricht das nicht genau der Idee der Ganzheitlichkeit des Xonbot-Ansatzes?

Nun, ein solcher Ansatz erinnert an das Verfahren eines Professors der berühmten Akademie von Lagado, die Jonathan Swifts Gulliver im Verlauf seiner Reisen aufgesucht hat.



Innerhalb eines quadratischen Rahmens von 20 Fuß Länge waren auf Holzstücken die einzelnen Wörter der Landessprache aufgetragen. Durch eine Drehung seitlich angebrachter Kurbeln konnte das ganze Wortsystem plötzlich geändert werden. Sinnvolle Wortkombinationen wurden abgeschrieben. Auf diese Weise wollte der Professor der Welt "einen vollständigen Inbegriff aller Künste und Wissenschaften" geben.

Das Problem ist das gleiche für den Xonbot: Die Komplexität wäre viel zu hoch. Das System müsste Ewigkeiten lernen. Und da sich das Nutzerverhalten zeitlich ändert würde es nie fertig werden.

Aus diesem Grunde müssen in der Realität verschiedene Aufgaben wie die Nutzersimulation, Energieoptimierung bzw. deren Teilaufgaben, Marketingautomatisierung oder Avatare getrennt modelliert und als separate Instanzen des Xonbot umgesetzt werden. Das ist auch deswegen sinnvoll, weil das Analysemodell des Xonbot - also die Aktionswertfunktion $f(s,a)$ - kaum menschlich interpretierbar ist. Damit ließen sich die einzelnen Funktionalitäten gar nicht separat evaluieren. In einer Analysewelt, in der alles mit allem verbunden wäre - der Fernseher die Energiesteuerung beeinflusst und die wiederum

die Beleuchtung und diese ihrerseits das Upselling und das wieder den Fernseher, etc. - könnte kein Mensch mehr erklären was eigentlich passiert. Fehlersuchen wären praktisch unmöglich. Darum müssen pragmatisch Kompromisse gefunden werden, indem die Smart-Home-Steuerung in Teilkomponenten zerlegt wird.

Einschub: Der Xonbot

Der Xonbot ist das Herzstück der Smart-Home-Automatisierungslösung der Signal Cruncher GmbH. Zwar vermag er der Welt keinen vollständigen Inbegriff aller mathematischen Künste und Theorien zu geben. Gleichwohl ist er ein Meilenstein in der praktischen Anwendung der künstlichen Intelligenz.

Herzstück des Xonbot ist sein Reinforcement-Learning-Framework. Dieses wurde ursprünglich von der prudsys AG - der Mutterfirma der Signal Cruncher - für den Bereich des Handels entwickelt.

Tafel der Besten

Die prudsys AG hat seit über zehn Jahren ein Reinforcement-Learning-Framework zunächst für den Bereich der Produktempfehlungen in Onlineshops entwickelt - das New Recommendation Framework (NRF). Im Laufe der Jahre wurde seine Anwendung immer weiter ausgeweitet. Im Ergebnis kann das NRF den Nutzern die richtigen Produkte zum richtigen Preis zur richtigen Zeit am richtigen Ort über den richtigen Kanal in Echtzeit so unterbreiten, dass ihr Kundenwert (wahlweise Umsatz oder Ertrag) maximiert wird. Damit vereint das NRF bisher getrennte Bereiche wie Personalisierung, Preisoptimierung und Mailingoptimierung in einer einheitlichen Applikation.

Die Aufgabe ist deswegen so herausfordernd, weil im Handel eine Vielzahl von Produkten existiert (meist zehntausende, im Buchbereich Millionen), von denen viele kaum eine Transaktionshistorie aufweisen. Also kaum gekauft, ja - im Internet - mitunter kaum angeklickt wurden! Diese Produkte werden auch als "Long Tail" bezeichnet und machen in den meisten Onlineshops eine Großteil des Produktsortiments aus. Nun erfordert RL aber viele Transaktionen, um robust zu arbeiten. Darum bedurfte es jahrelanger Forschung an Approximationsarchitekturen um diese Herausforderung zu meistern. Dabei spielt die Nutzung von Stammdaten (der Produkte, Nutzer, etc.) eine wichtige Rolle.

Der Ansatz des NRF erwies sich als so leistungsfähig, dass er universell verallgemeinert werden konnte und sein Framework komplett Handels-unabhängig ist. Das Ergebnis ist der Xonbot.

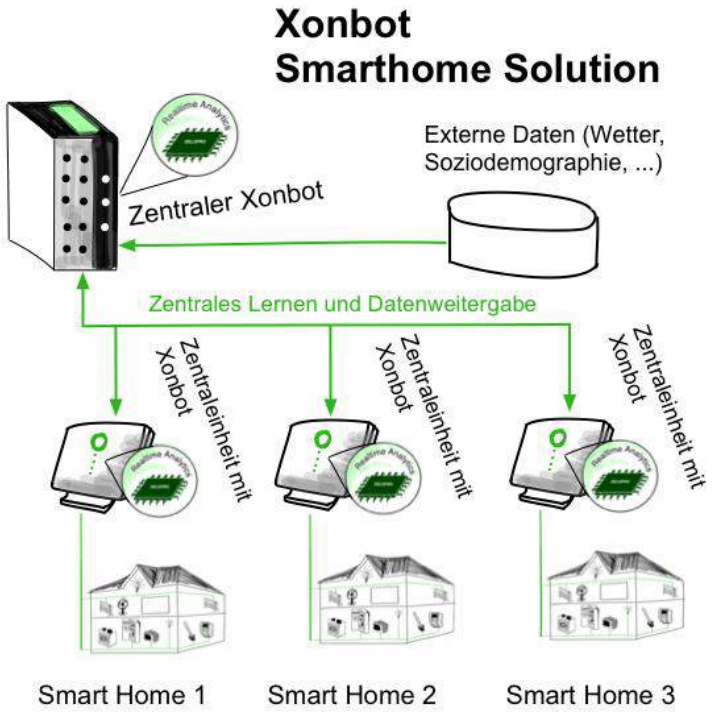
So universell der Xonbot jedoch auch ist, erfordert seine erfolgreiche Nutzung dessen geeignete Konfiguration. Das betrifft die Wahl der richtigen Attribute der Transaktions- und Stammdaten, geeignete Vorverarbeitung, passende Aggregation, Regressionsverfahren, usw. Das wiederum erfordert umfassendes Know-how über die Applikation.

Es soll an dieser Stelle auf die weitere Beschreibung des Xonbot verzichtet werden, da diese mathematisch sehr komplex ist und den Rahmen dieses Dokuments sprengen würde.

Einer für alle, alle für einen!

Der Xonbot kann zunächst autonom in jedem Haushalt arbeiten. Somit lernt er nur aus dem Verhalten des individuellen Haushalts, z.B. der Familie Meyer. In diesem Fall erfolgt die Integration des Xonbot z.B. direkt in der Zentraleinheit. Der Vorteil liegt in der Einfachheit der Lösung. Auch müssen Daten nicht per Internet weitergeleitet werden, was gerade im Zug der Datenschutzdiskussion von großem Vorteil ist. Alex ist beruhigt.

Jedoch wäre es von Vorteil, wenn ein Lernen auch haushaltsübergreifend stattfinden kann. Das ist insbesondere für die Modellierung des physikalischen Verhaltens bei der Minimierung des Energieverbrauchs hilfreich. Für Marketingapplikationen wie Upselling ist es sogar zwingende Voraussetzung. Ein solches Lernen ist durchaus möglich, indem eine zentrale Xonbot-Instanz aufgesetzt wird, welche aus den Daten aller Haushalte lernt (welche damit einverstanden sind).



Hier leiten alle lokalen Xonbots ihre Daten an die zentrale Lerninstanz weiter, welche ihnen das Analysemodell zurückliefert. Sofern die Verbindung zur Zentralinstanz unterbrochen wird, arbeiten die lokalen Xonbot-Instanzen autonom weiter. Dabei nutzen und aktualisieren sie das zuletzt von der Zentralinstanz erhaltene Analysemodell.

Sofern also Familie Meyer zustimmt, werden die Daten ihres Nutzungsverhaltens an einen zentralen Server zum Lernen geleitet, wohin auch die Daten der Nachbarn Herr Schmidt und Familie Popow fließen. Das Ganze kann wieder am Beispiel Schach illustriert werden: Der zentrale Xonbot entspricht einem Simultanspieler, der gleichzeitig an mehreren Brettern gegen die einzelnen Haushalte spielt und daraus lernt. Die einzelnen Partien - sprich Steuerung jedes einzelnen Haushalts - erfolgen gleichwohl individuell.

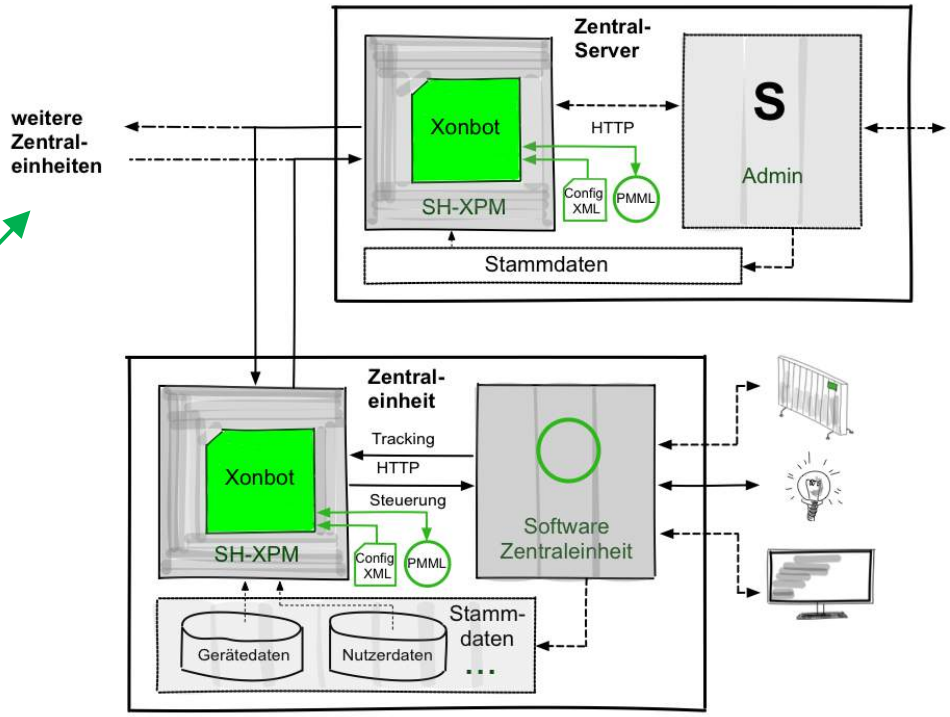
Einschub: Architektur der Lösung

In der Praxis erfolgt die Integration des Xonbot über einen Softwarecontainer, den SH-XPM, der die Integration des Systems deutlich vereinfacht.

Der SH-XPM (Smart-Home Xonbot-Manager) stellt eine einfach zu nutzende Software zur selbstlernenden Steuerung von Smart-Home-Devices dar. Er erweitert den Xonbot für diese Aufgabe. Am wirkungsvollsten geschieht das über die Anbindung einer Zentraleinheit - daher soll dieser Fall im weiteren erklärt werden. Gleichwohl kann an dieser Stelle ein beliebiges Gateway stehen, z.B. eine Bridge für Lampen wie bei Philips HUE.

Der SH-XPM ist eine Pure-Java-Applikation, die aus Kundenprojekten heraus entstanden ist und permanent weiterentwickelt wird. Er ist sehr schlank und kann unkompliziert erweitert werden. Der SH-XPM kann als Standalone-Programm genutzt oder als Java-Bibliothek in Drittapplikationen eingebunden werden.

Für den Fall der Integration in die Zentraleinheit (ZE) ergibt sich folgende Architektur:

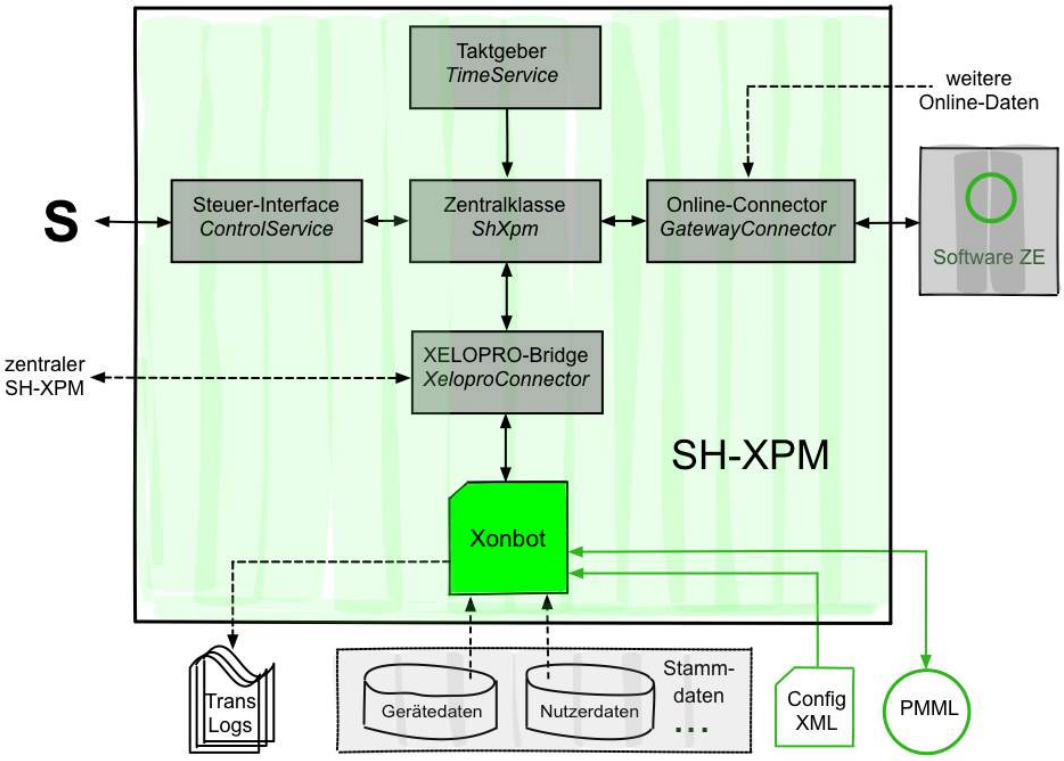


Sofern keine Zentralinstanz erwünscht ist, kann diese auch entfallen und jeder Haushalt lernt für sich allein.

Abschließend wollen wir kurz den Aufbau und die Arbeitsweise des SH-XPM vorstellen.

Die zentrale Instanz `ShXpm` führt alle relevanten Daten zusammen und steuert die Abläufe. Der Taktgeber gibt die Taktung entlang eines gleichbleibenden Zeitintervalls vor, definiert also unser *N*. Das Zeitintervall ist nutzerdefiniert und steht standardmäßig aus 5 Sekunden.

In jedem neuen Zeitpunkt nimmt der Online-Connector den aktuellen Zustand aller Geräte auf und steuert diese zugleich an. Die Anbindung an die Zentraleinheit geschieht über den Gateway-Connector. Dieser unterstützt diverse Kommunikationsprotokolle, z.B. MQTT und REST. Weitere Protokolle können leicht implementiert werden. Zusätzlich können noch externe Daten abgefragt werden, sofern diese nicht von der ZE bereitgestellt werden (Wetterdaten wie Sonnenauf- und Untergang, Bewegungsmelder, etc.).



Der aktuelle Gerätezustand wird nun als neuer Datensatz an den Xonbot durchgereicht, welcher sein Analysemodell aktualisiert. Nach dem Lernen wird der Xonbot umgekehrt nach dem nächsten Steuervorschlag der Geräte abgefragt. Dieser wird über den Online-Connector an die ZE weitergereicht, welche die entsprechenden Geräte ansteuert, deren Schaltung sich ändert.

Über das Steuer-Interface kann über wenige Befehle der Arbeitsmodus festgelegt werden:

Modus	Bedeutung
reset	Programm "vergisst" sein bisheriges Analysemodell.
learn	Programm lernt aus den Schaltvorgängen.
apply	Programm steuert die Geräte.
learnApply [Devicename]	Programm lernt und steuert zugleich [optional mit Gerätespezifikation].

Der Xonbot ist über den `XonbotConnector` angebunden, über den er durchgestartet wird und welcher dessen Online-Kommunikation durchstellt (insbesondere auch im Fall einer zentralen SH-XPM-Instanz). Die Konfiguration des Xonbot wird über seine XML-Konfigurationdatei definiert. Einige ihrer Settings werden jedoch vom `XonbotConnector` gemäß der Konfiguration des SH-XPM überschrieben. Auf diese Weise wird sichergestellt, dass mit einer festen Konfigurationsdatei gearbeitet werden kann, welche vom Nutzer nur in Spezialfällen angepasst werden muss.

Weiterhin kann der Xonbot auf externe Stammdaten zugreifen; entweder in Echtzeit (Datenbank) oder indem er sie beim Hochfahren in den Speicher lädt. Beim Runterfahren speichert der Xonbot sein Analysemodell als PMML-Datei, die er beim Hochfahren wieder einliest. PMML (Predictive Model Markup Language) ist ein XML-basierter Standard zum herstellerunabhängigen Austausch von Datenanalysemodellen. Durch diese Serialisierung geht kann das gelernte Wisse nicht verlorengehen.

Schließlich kann der Xonbot seine Transaktionsdaten auch in Logdateien speichern, den Transaktions-Logs, wodurch eine nachträgliche Simulation seines Lernverhaltens möglich wird.

Und jetzt das Ganze nochmal von vorn...

Die Idee hinter dem Xonbot besteht darin, die Schaltung von Smart-Home-Devices selbstlernend zu gestalten und mit zusätzlichen Services zu verknüpfen. Das vereinfacht zum einen die Bedienung, erhöht die Wirtschaftlichkeit und erlaubt ggf. zusätzlichen Umsatz für den Betreiber. Dabei werden in der Praxis jedoch meist Vorgaben gemacht - die Constraints; normalerweise in Form von Regeln. Das widerspricht der Philosophie des Xonbot nicht: Er darf im Rahmen dieser Regeln autonom agieren. Den Grad der Autonomie bestimmt der Nutzer selbst: Je restriktiver die Regeln, desto weniger Spielraum bleibt für die autonome Entscheidungsfindung, und umgekehrt.

Die zweite Idee hinter dem Xonbot ist seine Ganzheitlichkeit: Statt Spezialprobleme wie Energieverbrauchsoptimierung, Abwesenheitssimulation, personalisierte Unterhaltung oder Marketingautomation zu lösen, verknüpft er diese Ansätze miteinander und erlaubt damit völlig neue Einsatzszenarien, die in neue Geschäftsmodelle einmünden. Dazu setzt er auf ein einheitliches mathematisches Framework - das Reinforcement Learning - und eine einheitliche Echtzeit-Infrastruktur. In der Praxis erfordern die Use Cases allerdings oft doch eine Aufteilung auf separate Xonbot-Instanzen, da es aus Komplexitätsgründen illusorisch wäre alle Probleme mit einer einheitlich konfigurierten Instanz zu bearbeiten. Jedoch gestaltet sich die Einrichtung aufgrund der standardisierten IT-Infrastruktur des Xonbot zumeist einfach.

Aufgrund der Ganzheitlichkeit des Xonbot-Ansatzes kann dieser sein Potenzial natürlich am besten in Kombination mit einer Zentraleinheit ausspielen. Gleichwohl kann er auch für Spezialapplikationen wie Beleuchtungssteuerung, Energiemanagement, etc. eingesetzt werden.

Dem Prinzip des Reinforcement Learning folgend lernt der Xonbot in Echtzeit. Damit entfällt die mitunter aufwändige und Datenschutzrechtlich strittige Speicherung der Haushalt-Transaktionsdaten und das Analysemodell ist immer aktuell. Weiterhin arbeitet der Xonbot

für jeden Haushalt individuell, kann aber - sofern der Nutzer einverstanden ist - auch zentral lernen und damit die Qualität seiner Entscheidungsfindung verbessern sowie zusätzliche Services anbieten. Auch hier kann jeder Nutzer also wählen: zwischen erhöhtem Angebot und restriktiverer Datenhaltung.

Der Xonbot wird nicht direkt in Smart-Home-Applikationen eingebunden, da er ein generischer Analyseprozessor ist, der in verschiedensten Branchen genutzt wird. Stattdessen erfolgt die Integration über einen speziellen Softwarecontainer, den SH-XPM. Dieser ist in Java geschrieben und kann sowohl Standalone arbeiten als auch über seine Java-API aus Drittapplikationen heraus aufgerufen werden. Der Xonbot ist sehr schlank und kann im Zuge von Customizing schnell erweitert werden. Somit ist er ein hervorragendes Instrument, um in kurzer Zeit neue Ideen zu entwickeln und in Pilotprojekten auszuprobieren. Das Team von Signal Cruncher freut sich auf eine spannende Zusammenarbeit mit Ihnen!