Jun.-Prof. Dr. Christian Mendl Abgabe: 19. April 2018

Aufgabe 21 (Klassen in Python)
Ein grundlegender Baustein objektorientierter Programmiersprachen sind "Klassen"; diese legen die Datenstruktur eines Objekts fest. Das folgende Beispiel definiert eine neue Klasse `Person` in Python:

```
class Person(object):
    def __init__(self, first, last):
        self.firstname = first
        self.surname   = last
        self.hobbies = []   # initially empty list of hobbies

    def add_hobby(self, hobby):
        self.hobbies.append(hobby)
```

Methoden einer Klasse besitzen `self`[1] als erstes Argument, und die *Instanzvariablen* einer Klasse (`firstname`, `surname` und `hobbies` im Beispiel) werden durch Zuweisung (`self.firstname = ...`) definiert.

Die neue Klasse kann dann z.B. wie folgt eingesetzt werden:

```
sv = Person('Sebastian', 'Vettel')
sv.add_hobby('Motorsport')
sv.add_hobby('Football')
print(sv.hobbies)   # prints ['Motorsport', 'Football']
```

Sogenannte *Properties* sind spezielle Methoden, die wie eine Instanzvariable aufgerufen werden. Eine Ergänzung der obigen Klassendefinition um

```
    @property
    def fullname(self):
        return self.firstname + ' ' + self.surname
```

ermöglicht dann z.B. `print('The hobbies of', sv.fullname, 'are', sv.hobbies)`.

Klassen unterstützen *Vererbung*, d.h. die Definition einer abgeleiteten Klasse. Diese "erbt" die Instanzvariablen und Methoden der Basisklasse, kann diese aber überschreiben oder neue hinzufügen:

```
class Student(Person):
    def __init__(self, first, last, ID):
        super(Student, self).__init__(first, last)
        self.studentID = ID

    @property
    def fullname(self):
        return self.firstname + ' ' + self.surname + ' (student ID: ' + str(self.studentID) + ')'

stud = Student('Schlaubi', 'Schlumpf', 123)
stud.add_hobby('Reading')
print(stud.fullname)    # prints Schlaubi Schlumpf (student ID: 123)
```

Mittels `super(Student, self)` erhält man die Basisklasse.[2] Die Klasse `Student` hat die Methode `add_hobby` geerbt und die Methode `fullname` überschrieben ("method overriding").

(a) Erstellen Sie eine von `Person` abgeleitete Klasse `BankCustomer` mit einer zusätzlichen Instanzvariable `balance` (aktueller Kontostand). Definieren Sie Methoden `deposit(self, amount)` zum Einzahlen und `withdraw(self, amount)` zum Abheben, die jeweils den Kontostand aktualisieren und den neuen Kontostand zurückgeben. `withdraw` soll einen Fehler ausgeben, falls `amount` größer als der bisherige Kontostand ist. Testen Sie Ihre Implementierung anhand eines kleinen Beispiels. [2 Punkte]

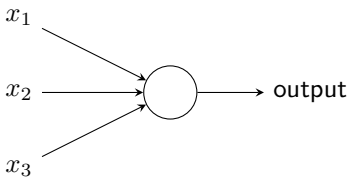
(b) Überschreiben Sie die Methode `add_hobby` in der Klasse `Student` mit neuer Signatur `add_hobby(self, hobby, preference)`. Diese soll das Tupel `(hobby, preference)` durch Aufruf von `add_hobby` der Basisklasse zur Liste der Hobbys hinzufügen. [1 Punkt]

(c) Erweitern Sie die obige Klasse `Person` um zwei Instanzvariablen `father` und `mother`. Erstellen Sie hiermit einen (fiktiven) Ahnenstammbaum einer Person bis zu den Großeltern. (Die Urgroßeltern können Sie auf `None` setzen.) Definieren Sie nun eine Methode `print_family_tree`, die den Stammbaum durch rekursive Aufrufe ausgibt. [3 Punkte]

[1] analog zum versteckten `this` Zeiger in C++

[2] In Python 3 genügt bereits `super()` ohne Argumente.

Aufgabe 22 (Perzeptron und NAND-Gate)
Das Perzeptron ist ein von Frank Rosenblatt 1958 vorgeschlagenes vereinfachtes Modell eines künstlichen neuronalen Netzwerks (ANN) und gilt als Vorläufer "moderner" ANNs. Für einen Eingabevektor $x \in \{0,1\}^d$ lautet die Ausgabe einer einzelnen Zelle mit Gewichten $w \in \mathbb{R}^d$ und Bias-Wert $b \in \mathbb{R}$

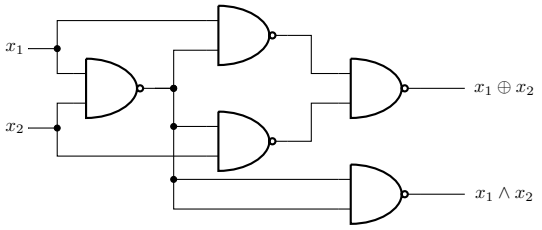


$$\text{output} = \begin{cases} 0, & w \cdot x + b \leq 0 \\ 1, & w \cdot x + b > 0 \end{cases}$$

wobei $w \cdot x$ das Skalarprodukt zwischen w und x bezeichnet.

(a) Rechnen Sie nach, dass für $d = 2$ und $w_1 = w_2 = -2$, $b = 3$ das logische NAND-Gate realisiert wird. [2 Punkte]

(b) Verifizieren Sie (z.B. durch Einsetzen aller 4 Kombinationen von $x_1, x_2 \in \{0,1\}$), dass nebenstehende Schaltung aus NAND-Gates tatsächlich $x_1 \oplus x_2$ und $x_1 \wedge x_2$ ausgibt, d.h. einen binären "Addierer" implementiert (mit $x_1 \wedge x_2$ der Übertrag der Addition). [2 Punkte]

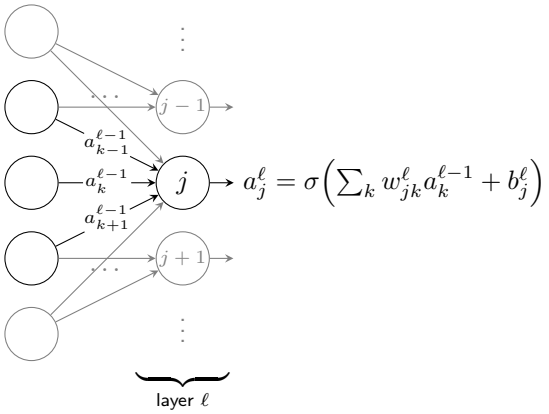


(c) Wie können w und b für allgemeines d gewählt werden, so dass die Perzeptron-Zelle die verallgemeinerte NAND-Operation $\neg(x_1 \wedge x_2 \wedge \cdots \wedge x_d)$ berechnet? [2 Punkte]

Tutoraufgabe (Backpropagation)
In einem Feedforward-künstlichen neuronalen Netz beträgt die Ausgabe der ℓ-ten Schicht

$$a^\ell = \sigma(z^\ell) \quad \text{mit} \quad z^\ell = w^\ell a^{\ell-1} + b^\ell,$$

wobei die "Aktivierungsfunktion" $\sigma : \mathbb{R} \to \mathbb{R}$ komponentenweise auf die Einträge von z^ℓ angewendet wird und w^ℓ die sogenannte Gewichts-Matrix und b^ℓ den Bias-Vektor der ℓ-ten Schicht bezeichnet. Es sei außerdem eine reellwertige *Kostenfunktion* C vorgegeben, die lediglich von der Ausgabe der letzten Schicht L abhängt: $C(a^L)$. Der Backpropagation-Algorithmus erlaubt die effiziente Berechnung der Ableitungen der Kostenfunktion bezüglich w^ℓ und b^ℓ. Im Folgenden verwenden wir zur Herleitung noch die Hilfsgröße



$$\delta_j^\ell := \frac{\partial C}{\partial z_j^\ell}.$$

(a) Zeigen Sie durch Anwenden der Kettenregel:

$$\delta^L = \operatorname{diag}\big(\sigma'(z^L)\big) \cdot \nabla_{a^L} C, \tag{BP1}$$

wobei die Ableitung σ' komponentenweise an den Einträgen von z^L ausgewertet wird und $\operatorname{diag}(d)$ die Diagonalmatrix mit Diagonale d bezeichnet.

(b) Leiten Sie (wiederum durch Anwenden der Kettenregel) den Zusammenhang

$$\delta^\ell = \operatorname{diag}\big(\sigma'(z^\ell)\big) \cdot \big(w^{\ell+1}\big)^T \cdot \delta^{\ell+1} \tag{BP2}$$

für alle $\ell < L$ her.

(c) Verifizieren Sie schließlich

$$\frac{\partial C}{\partial b_j^\ell} = \delta_j^\ell \tag{BP3}$$

sowie

$$\frac{\partial C}{\partial w_{jk}^\ell} = a_k^{\ell-1} \delta_j^\ell. \tag{BP4}$$