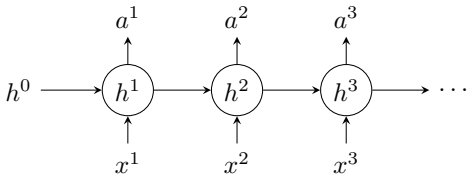
Aufgabe 31 (Manuelle Konstruktion eines Recurrent Neural Network)
Wir betrachten in dieser Aufgabe ein leicht modifiziertes Vanilla-RNN mit ReLU-Nichtlinearität, d.h.

$$h^t = \max(Wh^{t-1} + Ux^t + b, 0),$$
$$a^t = Vh^t$$

für $t = 1, 2, \ldots$, wobei das Maximum komponentenweise ausgewertet wird. Das RNN operiere auf den Buchstaben 'h', 'a', 'l', 'o', 'e', 'g' als Ein- und Ausgabe, die durch 'h'$= e_1$, 'a'$= e_2$, 'l'$= e_3$, 'o'$= e_4$, 'e'$= e_5$ und 'g'$= e_6$ kodiert seien (mit $e_i \in \mathbb{R}^6$ der i-te Einheitsvektor).



(a) Es sei zunächst $U = 0$ und $b = 0$, d.h. die Eingabe x^t spielt keine Rolle. Wählen Sie eine geeignete Dimension H für den hidden state h^t (also $h^t \in \mathbb{R}^H \ \forall t$), und finden Sie einen Anfangszustand h^0 sowie Matrizen $W \in \mathbb{R}^{H\times H}$ und $V \in \mathbb{R}^{6\times H}$, so dass die Ausgabesequenz "hallo" erzeugt wird, d.h. $a^1 = e_1$, $a^2 = e_2$, $a^3 = e_3$, $a^4 = e_3$, $a^5 = e_4$. [2 Punkte]

Hinweis: Unterscheiden Sie im hidden state zwischen dem ersten und zweiten 'l' in "hallo".

(b) Zeigen Sie, dass für $u, v \in \{0, 1\}$ und geeignete Wahl des Bias-Werts b der Ausdruck $\max(u + v + b, 0)$ gleich dem Produkt uv ist, d.h. das logische AND $u \wedge v$ berechnet wird. [1 Punkt]

(c) Realisieren Sie nun durch geeignete Wahl von W, U, b und V mit Anfangszustand $h^0 = 0$ eine einfache Variante der Wortvervollständigung: das Netzwerk soll bei Eingabe von $x^1 = e_1$, $x^2 = e_2$ und $x^t = 0$ $\forall t \geq 3$ die Ausgabesequenz "hallo" liefern, und bei Eingabe von $x^1 = e_1$, $x^2 = e_5$ und $x^t = 0$ $\forall t \geq 3$ die Ausgabesequenz "helga". Die korrekte Eingabe der ersten beiden Buchstaben 'h', 'a' bzw. 'h', 'e' sei zwingend erforderlich: z.B. darf die Eingabe $x^1 = e_2$, $x^t = 0$ $\forall t \geq 2$ nicht zu "allo" vervollständigt werden. [3 Punkte]

Aufgabe 32 (Recurrent Neural Network Implementierung)
In `char_rnn.zip` findet sich eine Python-Implementierung der Vanilla- und LSTM-Recurrent Neural Network-Architektur[1] mit Spezialisierung auf Buchstabensequenzen.

(a) Vervollständigen Sie den Code in `rnn_layers.py` an den (mit `#### ... ####`) angegebenen Stellen, und verifizieren Sie Ihre Lösung mittels `python -m unittest test_rnn_layers`. [3 Punkte]

Hinweis: Der Code berechnet in den Funktionen `rnn_step_forward` und `rnn_forward` bzw. `lstm_step_forward` und `lstm_forward` innerhalb `rnn_layers.py` noch nicht die lineare Transformation vom hidden state zur Ausgabe (d.h. $a^t = Vh^t$); dies wird später von der separaten Funktion `temporal_affine_forward` (inklusive einem zusätzlichen Bias-Vektor) umgesetzt.

(b) Die Klasse `CharRNN` in `char_rnn.py` realisiert ein konkretes RNN-Netzwerk, das auf Buchstabensequenzen (eines Textes) operiert. Machen Sie sich mit der Implementierung vertraut; wie wird der anfängliche "cell state" der LSTM-Zelle beim Generieren von Text (Buchstabe-für-Buchstabe Sampling) initialisiert? [1 Punkt]

(c) Lassen Sie nun `run_char_rnn.py` laufen, um ein RNN anhand der gesammelten Shakespeare-Werke in `shakespeare.txt` zu trainieren und gelegentlich Phantasie-Text zu erzeugen[2]. Geben Sie einen dieser generierten Textabschnitte an. [2 Punkte]

Hinweis: Es werden dabei noch keine vollständigen englischen Sätze herauskommen, aber zumindest gelegentliche Satzfragmente, die an Shakespeare erinnern. Das Programm läuft in einer Endlosschleife, d.h. Sie müssen es irgendwann manuell abbrechen.

[1] basierend auf http://cs231n.github.io/assignments2018/assignment3
[2] vergleiche mit http://karpathy.github.io/2015/05/21/rnn-effectiveness

Tutoraufgabe (Value-Iteration für die Bellman-Gleichung)
Wir untersuchen einen Markov Decision Process (MDP) mit endlichem Zustandsraum $\mathcal{S}$, möglichen Aktionen $\mathcal{A}$, Übergangswahrscheinlichkeit $\mathbb{P}(s_{t+1}|s_t, a_t)$ und Belohnungsfunktion $R : \mathcal{S} \to \mathbb{R}$. Die "Discounted Utility" $U^\pi(s_0)$ eines Anfangszustands $s_0 \in \mathcal{S}$ unter Policy-Funktion $\pi : \mathcal{S} \to \mathcal{A}$ ist definiert als

$$U^\pi(s_0) = \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t R(s_t)\Big|\pi\Big], \quad 0 < \gamma \leq 1,$$

wobei $\mathbb{E}$ den stochastischen Erwartungswert bezeichnet und "bedingt π" der Vorschrift $a_t = \pi(s_t)$ entspricht. Es sei $U = U^{\pi^*}$ die Utility-Funktion der optimalen (von s_0 unabhängigen) Policy π^*:

$$\pi^* = \mathrm{argmax}_\pi \, U^\pi(s_0).$$

Die *Bellman-Gleichung* für U besagt

$$U(s) = R(s) + \gamma \cdot \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) U(s').$$

Darauf aufbauend lautet der Value-Iteration-Algorithmus zur iterativen Bestimmung von U:

$$U_{i+1}(s) = R(s) + \gamma \cdot \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathbb{P}(s'|s, a) U_i(s') \quad \forall s \in \mathcal{S}.$$

Zeigen Sie für $\gamma < 1$, dass diese Gleichung eine Kontraktion bezüglich der Maximums-Norm

$$\|U\| = \max_{s \in \mathcal{S}} |U(s)|$$

ist und der Algorithmus somit stets zum eindeutigen Fixpunkt konvergiert. Warum ist dieser Algorithmus in der Praxis dennoch nicht empfehlenswert?