

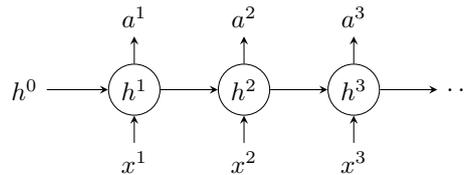
Aufgabe 19 (Manuelle Konstruktion eines Recurrent Neural Network)

Wir betrachten in dieser Aufgabe ein leicht modifiziertes Vanilla-RNN mit ReLU-Nichtlinearität, d.h.

$$h^t = \max(W h^{t-1} + U x^t + b, 0),$$

$$a^t = V h^t$$

für $t = 1, 2, \dots$, 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 20 (Recurrent Neural Network Implementierung)

In `char_rnn.zip` findet sich eine Python-Implementierung der Vanilla- und LSTM-Recurrent Neural Network-Architektur¹ mit Spezialisierung auf Buchstabensequenzen.

- (a) Vervollständigen Sie den Code in `rnn_layers.py` an den angegebenen Stellen, und verifizieren Sie Ihre Lösung mittels `python test_rnn_layers.py`. [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. $a^t = V h^t$); 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 erzeugen². 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.

¹basierend auf <http://cs231n.github.io/assignments2018/assignment3>

²vergleiche mit <http://karpathy.github.io/2015/05/21/rnn-effectiveness>

Tutoraufgabe 10 (Recurrent Neural Network in TensorFlow Keras)

Ein Netzwerkmodell mit zwei gestapelten LSTM-Schichten lässt sich mittels der TensorFlow Keras “Sequential API” realisieren als:

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=vocab_size),
    tf.keras.layers.LSTM(state_size, return_sequences=True, dropout=0.5),
    tf.keras.layers.LSTM(state_size, return_sequences=True, dropout=0.5),
    tf.keras.layers.TimeDistributed(
        tf.keras.layers.Dense(vocab_size, activation=tf.nn.softmax))
])
```

Die erwartete Eingabe besteht aus Zahlen aus der Menge $\{0, 1, \dots, \text{vocab_size} - 1\}$ und hat Dimension $N \times T$, wobei N die minibatch-Dimension ist und T die zeitliche Blocklänge. Diese Zahlen sind z.B. die Buchstaben-Indizes eines Textes, und werden von der Embedding-Schicht zunächst in entsprechende Einheitsvektoren der Länge vocab_size umgewandelt. Somit liefert die Embedding-Schicht einen Tensor mit Dimension $N \times T \times \text{vocab_size}$.

Eine RNN-Schicht `tf.keras.layers.LSTM` erhält also ganze zeitliche Sequenzabschnitte auf einmal, analog zur Funktion `lstm_forward` in `rnn_layers.py` aus A20. Der Parameter `state_size` gibt die Dimension des hidden state an.

Schließlich wendet die Ausgabeschicht `TimeDistributed` eine Operation zu jedem Zeitschritt an, ähnlich zur Funktion `temporal_affine_forward` in `rnn_layers.py`.

Trainieren Sie das Modell anhand der Shakespeare-Werke wie in A20 (c), und generieren Sie anschließend wiederum Textabschnitte Buchstabe für Buchstabe.

Hinweis: `model.predict(x)` liefert die Ausgabe des Modells für Eingabe x .