Recurrent Neural Networks (RNNs)

Dr. Benjamin Roth, Nina Poerner

CIS LMU München

Heute

- 9:15 - 10:45: RNN Basics + CNN
- 11:00 - 11:45: Übung PyTorch
- Statt Übungsblatt bis nächste Woche durcharbeiten:
  - http://www.deeplearningbook.org/contents/rnn.html (Abschnitte 10.0 - 10.2.1, 10.7, 10.10)
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Nächste Woche:
  - 9:15 - 10:00: "Journal Club" zu LSTM
  - 10:00 - 10:45: Keras (Teil 2)
  - 11:00 - 11:45: Übung Word2Vec

Recurrent Neural Networks (RNNs)

- Family of neural networks for processing sequential data $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(T)}$.
- Sequences of words, characters, ...
- Simplest case: for each time step t, compute representation $\mathbf{h}^{(t)}$ from current input $\mathbf{x}^{(t)}$ and previous representation $\mathbf{h}^{(t-1)}$.

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$$

- $\mathbf{x}^{(t)}$ can be embeddings, one-hot, output of some previous layer ...
- **Question:** By recursion, what does $\mathbf{h}^{(t)}$ depend on?
  - all previous inputs $\mathbf{x}^{(1)} \ldots \mathbf{x}^{(t)}$
  - the initial state $\mathbf{h}^{(0)}$ (typically all-zero, but not necessarily, c.f. encoder-decoder)
  - the parameters of θ

Parameter Sharing

- Going from a time step $t-1$ to t is parameterized by the same parameters θ for all t!

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$$

- **Question:** Why is parameter sharing a good idea?
  - Fewer parameters
  - Can learn to detect features regardless of their position
  - Can generalize to longer sequences than were seen in training

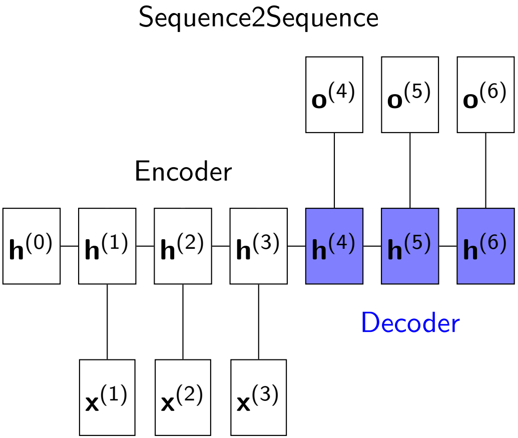
RNN: Output

- The output at time t is computed from the hidden representation at time t:

$$\mathbf{o}^{(t)} = f(\mathbf{h}^{(t)}; \mathbf{V}_o)$$

- Typically a linear transformation: $\mathbf{o}^{(t)} = \mathbf{V}_o^T \mathbf{h}^{(t)}$
- Some RNNs compute $\mathbf{o}^{(t)}$ at every time step, others only at the last time step $\mathbf{o}^{(T)}$

RNN: Output



Machine Translation, Summarization, Inflection, ...

Any questions so far?

RNN: Loss Function

- Loss function:
  - Several time steps: $\mathcal{L}(y^{(1)}, \dots y^{(T)}; \mathbf{o}^{(1)} \dots \mathbf{o}^{(T)})$
  - Last time step: $\mathcal{L}(y; \mathbf{o}^{(T)})$
- Example: POS Tagging
  - Output $\mathbf{o}^{(t)}$ is predicted distribution over POS tags
    - ⋆ $\mathbf{o}^{(t)} = P(\text{tag} =?|\mathbf{h}^{(t)})$
    - ⋆ Typically: $\mathbf{o}^{(t)} = \mathrm{softmax}(\mathbf{V}_o^T \mathbf{h}^{(t)})$
  - Loss at time t: negative log-likelihood (NLL) of true label $y^{(t)}$

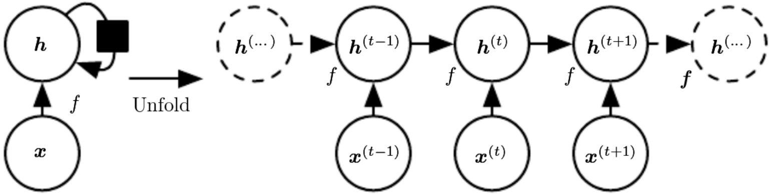
$$\mathcal{L}^{(t)} = -\log P(\text{tag} = y^{(t)}|\mathbf{h}^{(t)}; \mathbf{V}_o)$$

  - Overall Loss for all time steps:

$$\mathcal{L} = \sum_{t=1}^{T} \mathcal{L}^{(t)}$$

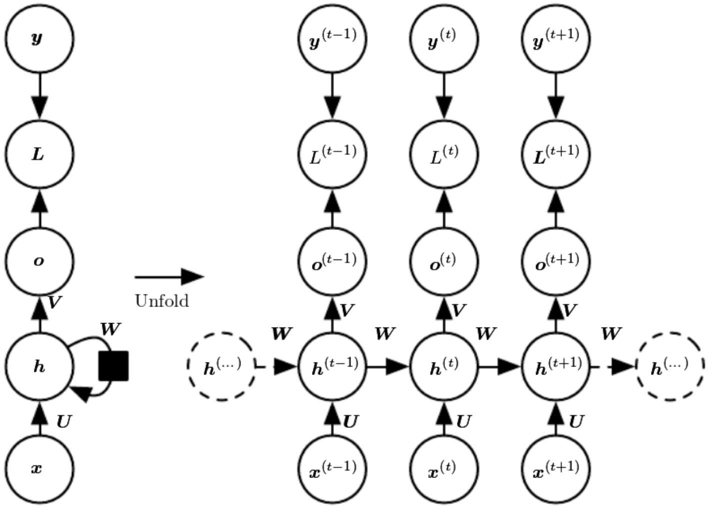
Graphical Notation

- Nodes indicate input data (**x**) or function outputs (otherwise).
- Arrows indicate functions arguments.
- Compact notation (left):
  - All time steps conflated.
  - ■ indicates *"delay"* of 1 time unit.



Source: Goodfellow et al.: Deep Learning.

Graphical Notation: Including Output and Loss Function



Source: Goodfellow et al.: Deep Learning.

Any questions so far?

Backpropagation through time

- We have calculated loss $\mathcal{L}$ at time step T and want to update θ with gradient descent.
- For now, imagine that we have time step-specific "dummy"-parameters $\theta^{(t)}$, which are identical copies of θ
- $\rightarrow$ the unrolled RNN looks like a feed-forward-neural-network!
- $\rightarrow$ we can calculate $\frac{\partial \mathcal{L}}{\partial \theta^{(t)}}$ using standard backpropagation
- **Question:** How to calculate $\frac{\partial \mathcal{L}}{\partial \theta}$?
- Add up the "dummy" gradients:

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial \theta^{(t)}}$$

Truncated backpropagation through time

- Simple idea: Stop backpropagation through time after k time steps

$$\frac{\partial \mathcal{L}}{\partial \theta} = \sum_{t=T-k}^{T} \frac{\partial \mathcal{L}}{\partial \theta^{(t)}}$$

- **Question:** What are advantages and disadvantages?
  - Advantage: Faster and parallelizable
  - Disadvantage: If k is too small, long-range dependencies are hard to learn

Any questions so far?

Vanilla RNN

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta) = \tanh(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(h-1)} + \mathbf{b})$$

$$\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{b}\}$$

- **W**: Hidden-to-hidden
- **U**: Input-to-hidden
- **b**: Bias term
- Vanilla RNN in keras:

```
vanilla = SimpleRNN(units=10, use_bias = True)
vanilla.build(input_shape = (None, None, 30))
print([weight.shape for weight in vanilla.get_weights()])
[(30, 10), (10, 10), (10,)]
```

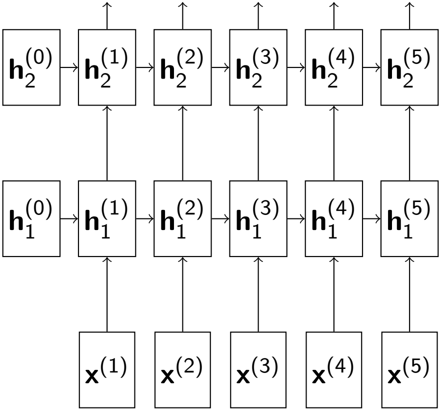
- **Question:** Which shape belongs to which weight?

Bidirectional RNNs

- Conceptually: Two RNNs that run in opposite directions over the same input
- Typically, each RNN has its own set of parameters
- Two sequences of hidden vectors: $\overrightarrow{\mathbf{h}}^{(1)} \dots \overrightarrow{\mathbf{h}}^{(T)}$, $\overleftarrow{\mathbf{h}}^{(1)} \dots \overleftarrow{\mathbf{h}}^{(T)}$
- Typically, $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$ are concatenated along their hidden dimension
- **Question:** Which hidden vectors should we concatenate if our output layer needs a single hidden vector $\mathbf{h}$?
  - $\mathbf{h} = \overrightarrow{\mathbf{h}}^{(T)} || \overleftarrow{\mathbf{h}}^{(1)}$
  - Because these are the vectors that have "read" the entire sequence
- **Question:** Which hidden vectors should we concatenate if we need one hidden vector per time step t?
  - $\mathbf{h}^{(t)} = \overrightarrow{\mathbf{h}}^{(t)} || \overleftarrow{\mathbf{h}}^{(t)}$
  - Full left context, full right context

Multi-Layer RNNs

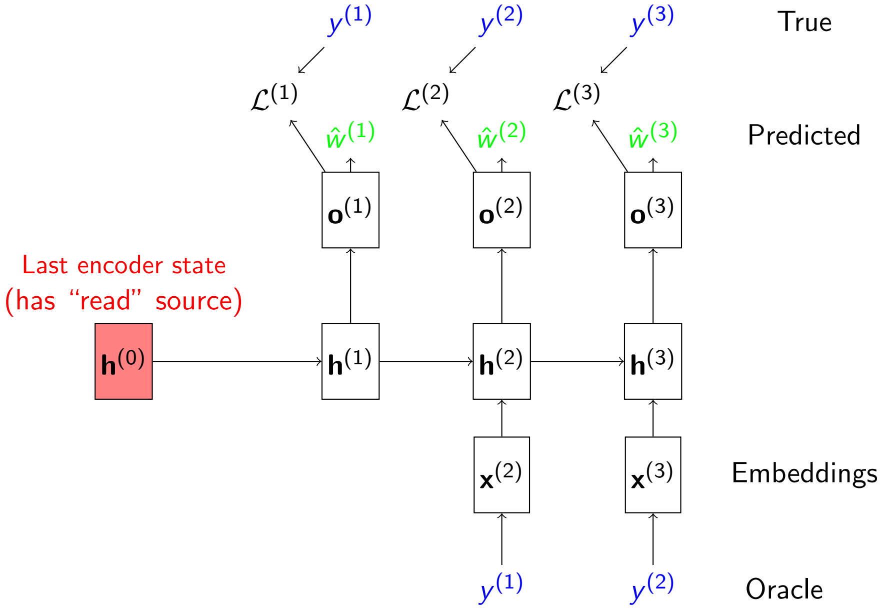
- Conceptually: A stack of L RNNs, such that $\mathbf{x}_l^{(t)} = \mathbf{h}_{l-1}^{(t)}$.



Feeding outputs back

- What do we do if the input sequence $\mathbf{x}^{(1)} \dots \mathbf{x}^{(T)}$ is only given at training time, but not at test time?
- Examples: Machine Translation decoder, (generative) language model

Example: Machine Translation



Oracle signal

- Give Neural Network a signal that it will not have at test time
- Can be useful during training (e.g., mix oracle and predicted signal)
- Can establish upper bounds of modules

Gated RNNs: Teaser

- Vanilla RNNs are not frequently used, as they tend to forget past information quickly
- Instead: LSTM, GRU, ... (next week!)