Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

Recurrent Neural Networks

Frank Hutter Abhinav Valada

University of Freiburg

Overview of Week 7

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

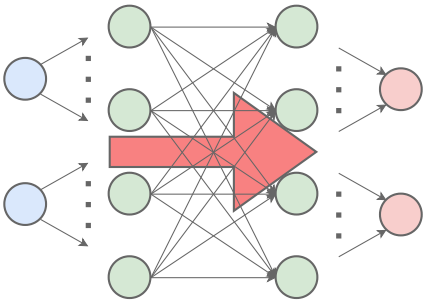
Week 7: Recurrent Neural Networks

Introduction

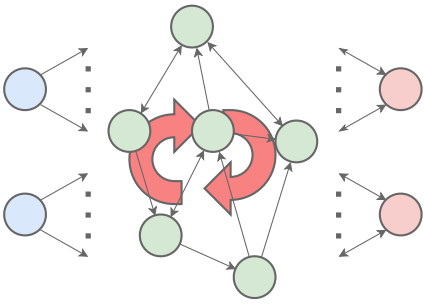
Frank Hutter Abhinav Valada

University of Freiburg

Feedforward vs Recurrent Neural Networks



Feedforward Network



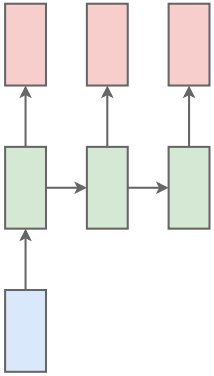
Recurrent Network

Recurrent Neural Networks (RNNs)

- Neural Networks that allow for cycles in the connectivity graph.
- Cycles let information persist in the network for some time (state), and provide a time-context or (fading) memory.
- Very powerful for processing sequences.
- Implement dynamical systems rather than function mappings, and can approximate any dynamical system with arbitrary precision.
- They are Turing-complete [Siegelmann and Sontag, 1991].

Sequence to Sequence Mapping – One-to-Many

one-to-many



e.g. image caption generation

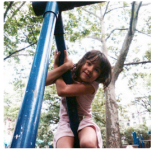
"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

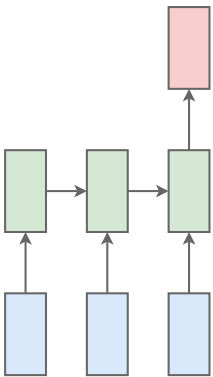


"young girl in pink shirt is swinging on swing."

[Karpathy and Fei-Fei, 2015]

Sequence to Sequence Mapping – Many-to-One

many-to-one

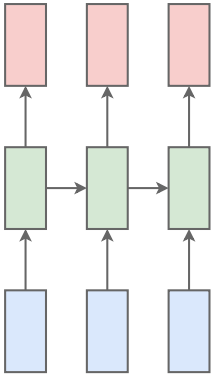


e.g. sentiment classification

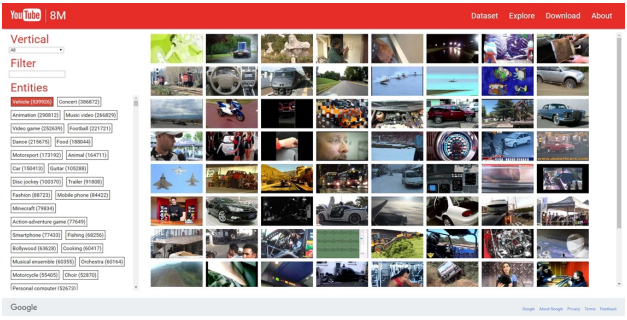
Review	Rating
"The movie is great! I really like it because it is so good!"	★★★★☆
"Did not really fit my taste, will not recommend it."	★★☆☆☆
"This movie sucks! Give me back my money."	★☆☆☆☆

Sequence to Sequence Mapping – Many-to-Many

many-to-many



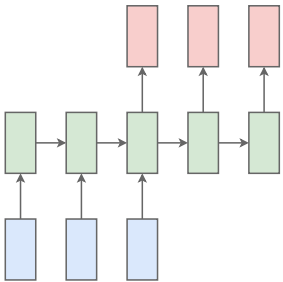
e.g. video frame classification



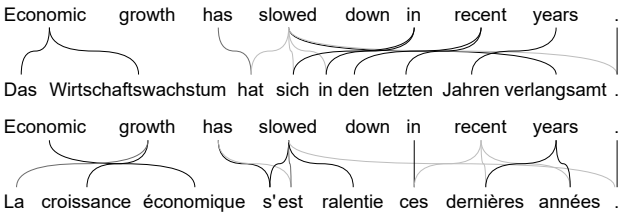
[Abu-El-Haija et al., 2016]

Sequence to Sequence Mapping – Many-to-Many (cont.)

many-to-many



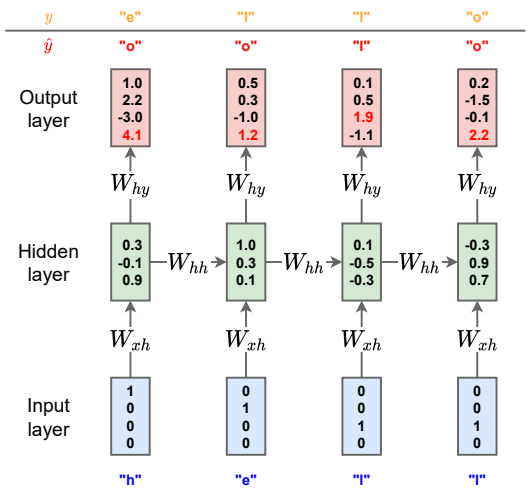
e.g. machine translation



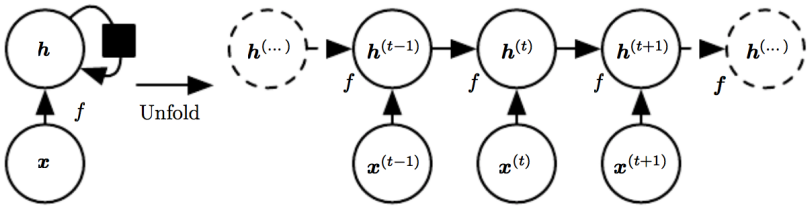
Example: Neural Language Model

Character-level language model:

- Vocabulary: [h, e, l, o]
- Training sequence: “hello”



Unfolding the Computational Graph of an RNN



[Image source: Goodfellow et al., 2016, p. 370, Fig. 10.2]

$$\begin{aligned} \boldsymbol{h}^{(t)} &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \\ &= f(f(\boldsymbol{h}^{(t-2)}, \boldsymbol{x}^{(t-1)}; \boldsymbol{\theta}), \boldsymbol{x}^{(t)}; \boldsymbol{\theta}) \end{aligned}$$

Regardless of the sequence length, the learned model always has the same input size!

Questions to Answer for Yourself / Discuss with Friends

- Relation to your interests: Can you think of other applications that we did not discuss where RNNs are used?
- Application of what you just learned: Can we use RNNs to learn from non-sequential data, e.g., from an image? If so what would be the benefit?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

RNN Design Patterns

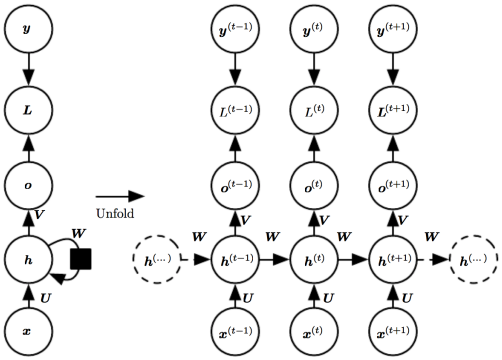
Frank Hutter Abhinav Valada

University of Freiburg

Important Design Patterns for RNNs

- Outputs at every step, recurrent connections between hidden units.
- Trained with Backpropagation Through Time (BPTT) $\rightarrow$ backprop with weight sharing over the unrolled computational graph.

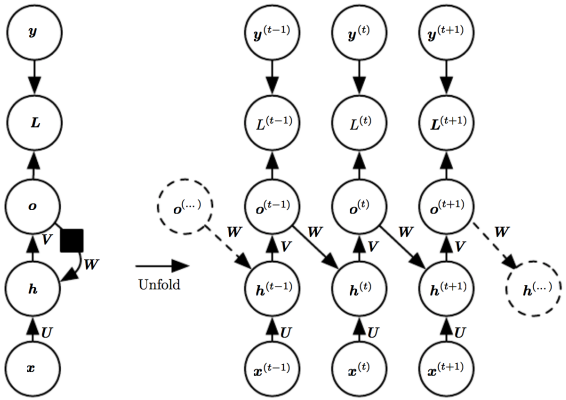
$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$



[Image source: Goodfellow et al., 2016, p. 373, Fig. 10.3]

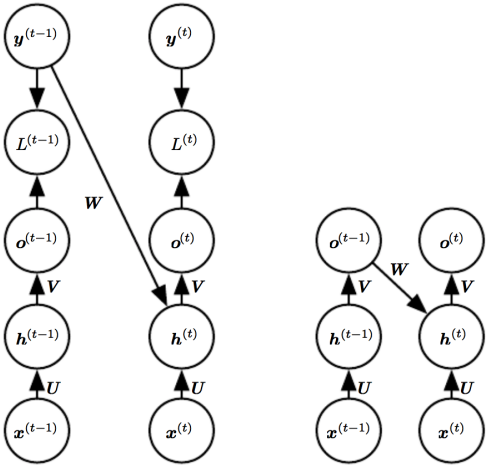
Important Design Patterns for RNNs (cont.)

- Recurrent connections from output to hidden units at next step.
- Potentially less powerful, but can be trained with teacher forcing instead of BPTT.



[Image source: Goodfellow et al., 2016, p. 375, Fig. 10.4]

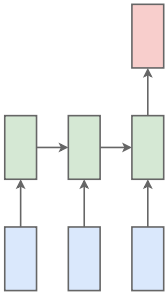
Teacher Forcing

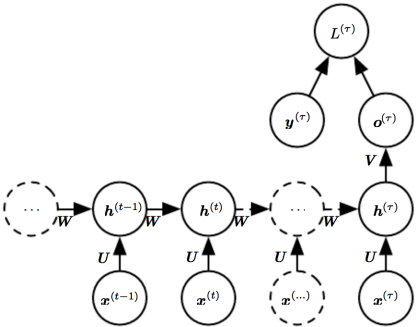


[Image source: Goodfellow et al., 2016, p. 377, Fig. 10.6]

Important Design Patterns for RNNs (cont.)

- Recurrent connections between hidden units, output only at the last step (to produce a summary of the sequence).
- Trained with BPTT.

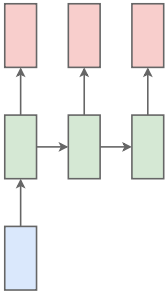


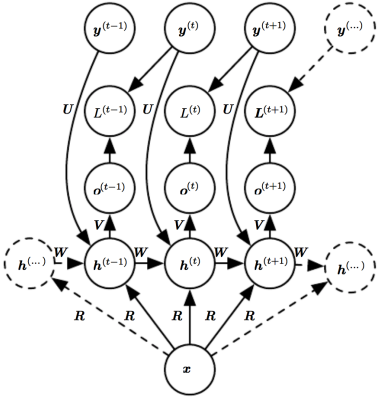


[Image source: Goodfellow et al., 2016, p. 376, Fig. 10.5]

Modeling Sequences Conditioned on Context

- Mapping static input $\boldsymbol{x}$ into distribution over sequences of $\boldsymbol{y}$, e.g., image captioning.

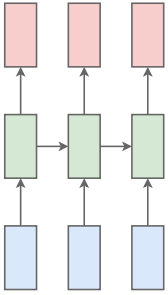


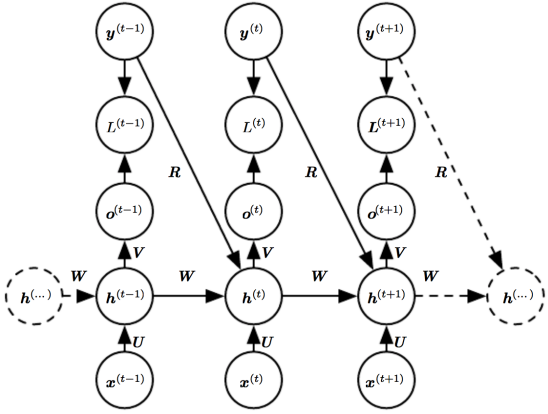


[Image source: Goodfellow et al., 2016, p. 386, Fig. 10.9]

Modeling Sequences Conditioned on Context (cont.)

- Mapping variable-length sequence of inputs $\boldsymbol{x}$ into distribution over sequences of $\boldsymbol{y}$.

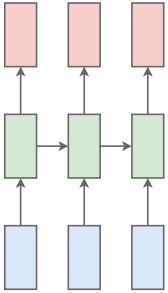


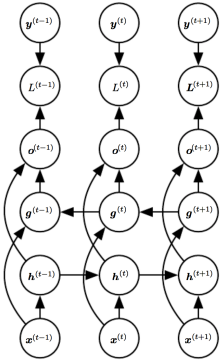


[Image source: Goodfellow et al., 2016, p. 387, Fig. 10.10]

Bidirectional RNNs

- Two separate streams of recurrent information, summarizing information from the past and the future.

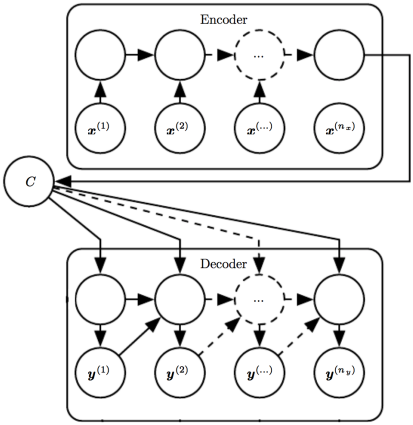




[Image source: Goodfellow et al., 2016, p. 389, Fig. 10.11]

Encoder-Decoder Sequence-to-Sequence Architectures

- Input and output sequence may be of different length, e.g., in machine translation.



[Image source: Goodfellow et al., 2016, p. 391, Fig. 10.12]

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What are the advantages and disadvantages of training using teacher forcing?
- Application of what you just learned: Can we use Bidirectional RNNs for a handwriting recognition task?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

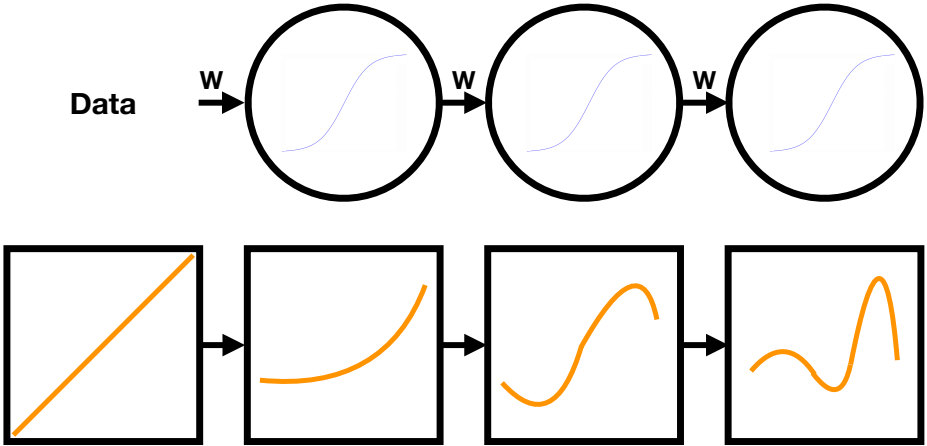
Week 7: Recurrent Neural Networks

Difficulties in Training RNNs

Frank Hutter Abhinav Valada

University of Freiburg

Repeated Function Composition



- Repeated application of the same function leads to powerful non-linear processing but also creates problems.

Repeated Function Composition (cont.)

- Consider recurrence relation without non-linearity

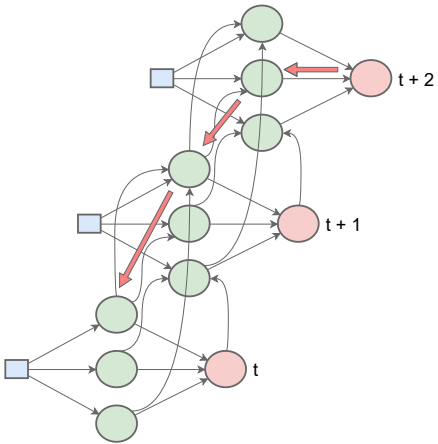
$$\boldsymbol{h}^{(t)} = \boldsymbol{W}^\top \boldsymbol{h}^{(t-1)} = (\boldsymbol{W}^t)^\top \boldsymbol{h}^{(0)}$$

- Eigen-Decomposition of weight matrix:

$$W = Q \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix} Q^{-1} \implies W^t = Q \begin{pmatrix} \lambda_1^t & & \\ & \ddots & \\ & & \lambda_n^t \end{pmatrix} Q^{-1}$$

- $\lambda_i < 1$: component vanishes, $\lambda_i > 1$: component explodes

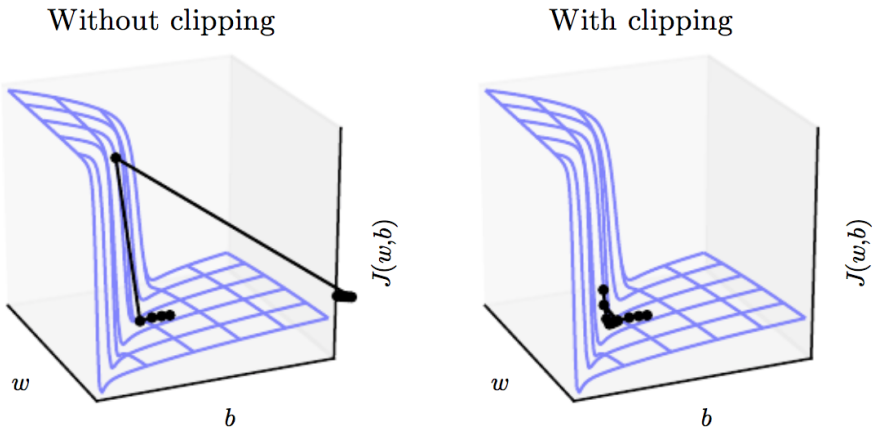
Vanishing and Exploding Gradients



- Network unrolled in time for training, e.g., in BPTT.
- Error gradient is propagated back and multiplied with weight w and derivative of unit's activity $f'(net)$.
- If $|f'(net)w > 1|, \forall t$
  exponential increase $\Rightarrow$ oscillating weights
- If $|f'(net)w < 1|, \forall t$
  exponential decrease $\Rightarrow$ slow convergence

Gradient Clipping

- To combat exploding gradients, clip gradients at each time step.



[Image source: Goodfellow et al., 2016, p. 409, Fig. 10.17]

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What are the two most common problems that can affect the training of RNNs?
- Repetition: What are some solutions to alleviate these problems?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

Backpropagation for RNNs

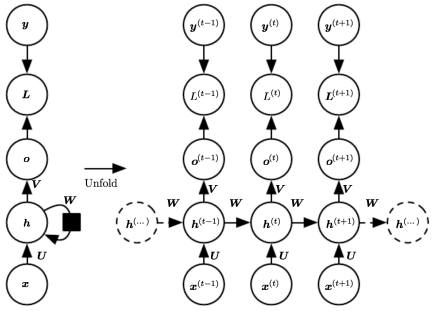
Frank Hutter Abhinav Valada

University of Freiburg

Backpropagation Through Time (BPTT)

- Backprop with weight sharing over the unrolled computational graph.
- Example with $\tanh$ nonlinearities, $\mathrm{softmax}$ output, and negative log-likelihood of the outputs at the target class.
- Trainable parameters are $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{b}$, and $\boldsymbol{c}$.

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)} \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)} \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)})
\end{aligned}
$$



[Image source: Goodfellow et al., 2016, p. 373, Fig. 10.3]

Backpropagation for RNN

As before, we compute the gradient $\nabla_{\boldsymbol{N}} L$ recursively for every node $\boldsymbol{N}$. Start with the node immediately preceding the final loss:

$$\frac{\partial L}{\partial L^{(t)}} = 1$$

Gradient $\nabla_{\boldsymbol{o}^{(t)}} L$ of outputs at time t $(t = 1 \dots \tau)$, for all i, t:

$$\left(\nabla_{\boldsymbol{o}^{(t)}} L\right)_i = \frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i^{(t)}} = \hat{y}_i^{(t)} - \mathbf{1}_{i=y^{(t)}}$$

Recap: Chain Rule Beyond Scalars

As a generalization of the scalar case, consider $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{y} \in \mathbb{R}^n$, $g : \mathbb{R}^m \to \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$. If $\mathbf{y} = g(\mathbf{x})$ and $z = f(\mathbf{y})$, then

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

We can write this in a more compact way using vector notation as:

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^{\mathsf{T}} \nabla_{\mathbf{y}} z$$

where $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ is the $n \times m$ Jacobian matrix of g, and with $\nabla_{\mathbf{y}} z$ being the gradient of z with respect to the vector $\mathbf{y}$. In Backpropagation, this is applied systematically to calculate the gradient of the loss L w.r.t. some node $\boldsymbol{p}$ parent to a descendant node $\boldsymbol{d}$:

$$\nabla_{\mathbf{p}} L = \left(\frac{\partial \mathbf{d}}{\partial \mathbf{p}}\right)^{\mathsf{T}} \nabla_{\mathbf{d}} L$$

Backpropagation for RNN (cont.)

We work our way backwards, starting from the last time step τ:

$$\nabla_{\boldsymbol{h}^{(\tau)}} L = \left(\frac{\partial \boldsymbol{o}^{(\tau)}}{\partial \boldsymbol{h}^{(\tau)}} \right)^{\top} \nabla_{\boldsymbol{o}^{(\tau)}} L = \boldsymbol{V}^{\top} \nabla_{\boldsymbol{o}^{(\tau)}} L$$

We then iterate backwards to backprop the gradients through time for $t = \tau - 1 \ldots 1$. For these time steps, $\boldsymbol{v}^{(t)}$ has two descendants, $\boldsymbol{o}^{(t)}$ and $\boldsymbol{h}^{(t+1)}$, thus the gradient is calculated as:

$$\begin{aligned} \nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}} \right)^{\top} (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}} \right)^{\top} (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^{\top} \operatorname{diag} \left(1 - \left(\boldsymbol{h}^{(t+1)} \right)^2 \right) (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^{\top} (\nabla_{\boldsymbol{o}^{(t)}} L) \end{aligned}$$

Backpropagation for RNN (cont.)

Once gradients of internal nodes are computed, we can obtain gradients on the parameter nodes:

$$\nabla_{\boldsymbol{c}} L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}} \right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L$$

$$\nabla_{\boldsymbol{b}} L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}} \right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)} \right)^2 \right) \nabla_{\boldsymbol{h}^{(t)}} L$$

$$\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}} \right) \nabla_{\boldsymbol{V}^{(t)}} o_i^{(t)} = \sum_t \left(\nabla_{\boldsymbol{o}^{(t)}} L \right) \boldsymbol{h}^{(t)\top}$$

Backpropagation for RNN (cont.)

$$\begin{aligned}\nabla_{\boldsymbol{W}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{W}} h_i^{(t)} \\ &= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{h}^{(t-1)\top}\end{aligned}$$

$$\begin{aligned}\nabla_{\boldsymbol{U}} L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\boldsymbol{U}^{(t)}} h_i^{(t)} \\ &= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \left(\nabla_{\boldsymbol{h}^{(t)}} L\right) \boldsymbol{x}^{(t)\top}\end{aligned}$$

BPTT in Practice

Problem: The propagation of the gradient through all previous nodes $h^{(t)}$ (of the unrolled graph) is expensive and potentially unstable (vanishing/exploding gradients).
$\Rightarrow$ Solution: Truncated BPTT

- Forward pass: compute for the entire graph.
- Backward pass: truncate the gradient after fixed intervals (usually aligned with mini batches).
- Thus, use sequential data loading during training.

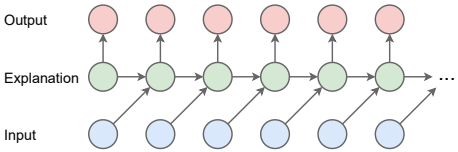


Figure: Standard BPTT.

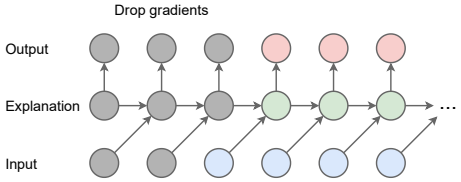


Figure: Truncated BPTT.

Questions to Answer for Yourself / Discuss with Friends

- Repetition: How does Backpropagation through time work intuitively?
- Discuss: What are differences to regular Backprop discussed before?
- Discuss: Could we also just use the gradient of the loss w.r.t the network output at each time step to train an RNN, basically ignoring any recurrence? What do you think would be the consequence?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

LSTM: Long Short-Term Memory

Frank Hutter Abhinav Valada

University of Freiburg

Recap: BPTT Equations and Vanishing/Exploding Gradients

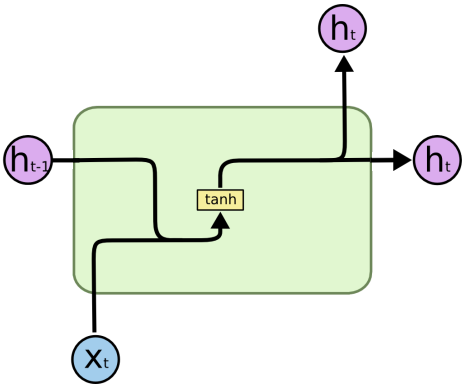
When calculating all the $\nabla_{\boldsymbol{h}^{(t)}} L$ terms in BPTT, the Jacobians $\left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top$ carry the gradients backwards in time through the recurrent chain:

$$\begin{aligned}\nabla_{\boldsymbol{h}^{(t)}} L &= \left(\frac{\partial \boldsymbol{h}^{(t+1)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h}^{(t)}}\right)^\top (\nabla_{\boldsymbol{o}^{(t)}} L) \\ &= \boldsymbol{W}^\top \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t+1)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t+1)}} L) + \boldsymbol{V}^\top (\nabla_{\boldsymbol{o}^{(t)}} L)\end{aligned}$$

Multiplying with $\boldsymbol{W}^\top \operatorname{diag}\left(1 - (\boldsymbol{h})^2\right)$ repeatedly, if the Eigenvalues of this matrix are < 1, we will likely get vanishing gradients; if they are > 1, gradients will likely explode.

New Visualization of an RNN Unit

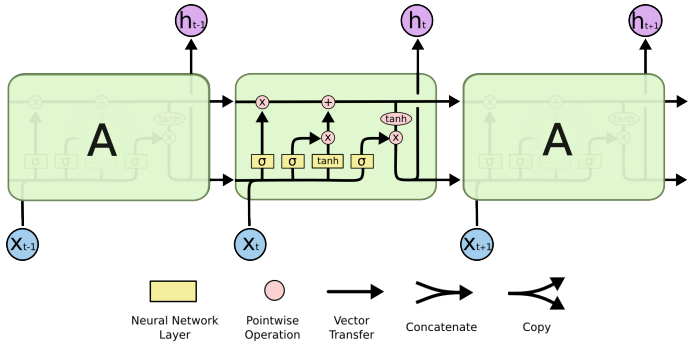
Visualization of one unit in a regular or *vanilla* RNN



[Image source: Olah, 2015 (adapted and reproduced with permission)]

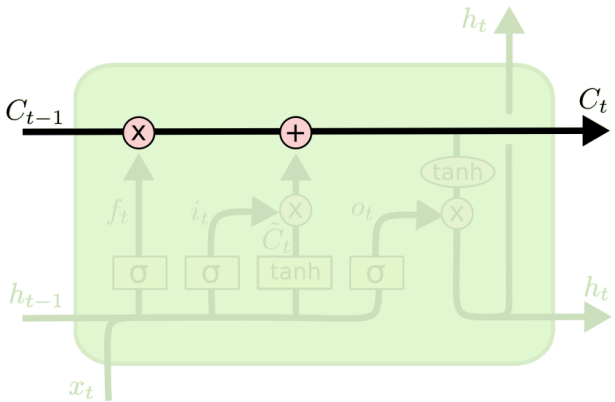
Long Short-Term Memory (LSTM) Networks

- Introduced in [Hochreiter and Schmidhuber, 1997].
- Address the vanishing gradient problem through units with special structure.
- Memory cell with forget, input and output gates.
- By now: standard RNN model in many applications.



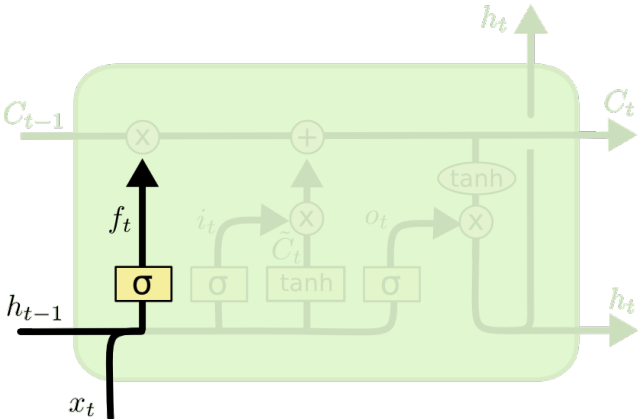
[Image source: Olah, 2015 (reproduced with permission)]

LSTM – Cell State



[Image source: Olah, 2015 (reproduced with permission)]

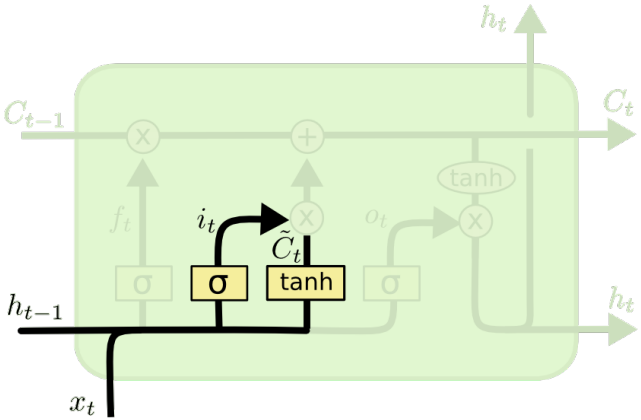
LSTM – Forget Gate



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

[Image source: Olah, 2015 (reproduced with permission)]

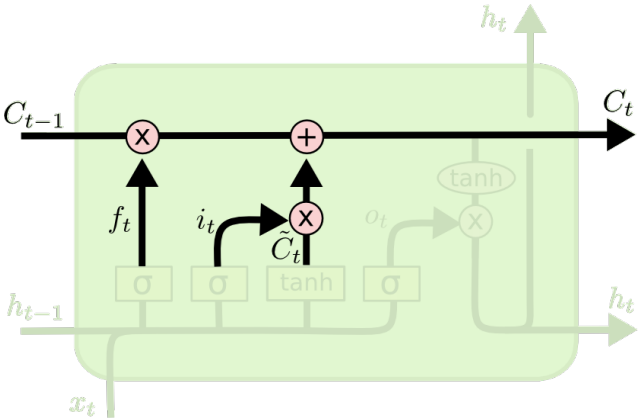
LSTM – Input Gate and Candidate Value



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

[Image source: Olah, 2015 (reproduced with permission)]

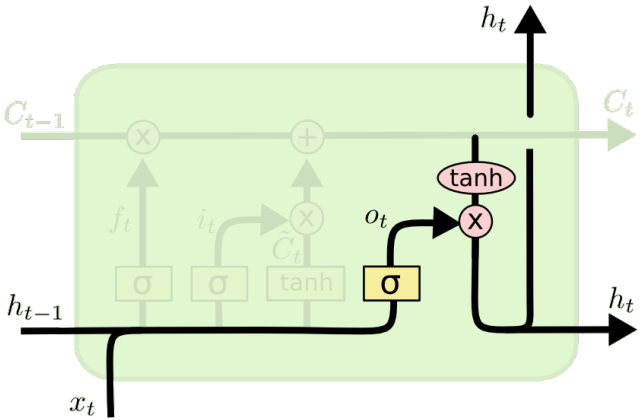
LSTM – Cell Update



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$

[Image source: Olah, 2015 (reproduced with permission)]

LSTM – Output Gate and Hidden State



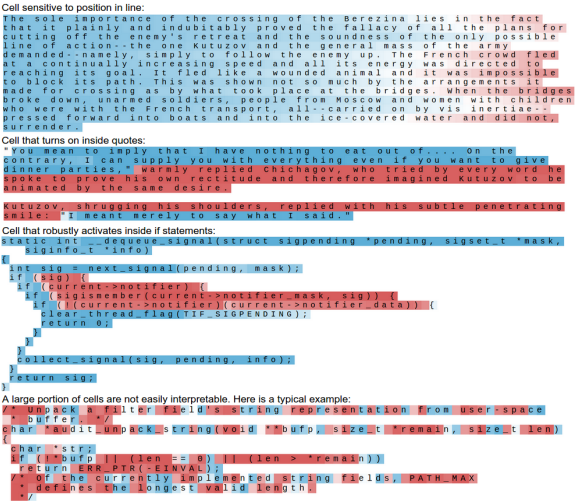
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

[Image source: Olah, 2015 (reproduced with permission)]

Power of the Gating Mechanism

- LSTM can make variables *internal*, just for further computation.
- LSTM can extract features of current input and add them selectively to its state.
- LSTM can selectively forget and remember features.
- LSTM variables already have their meaning:
  less training and less variables needed.
- LSTM weights can more easily be interpreted.
- Fewer gradient decay problems because information flows by default.

Visualizing Features Learned by LSTM Units



[Image source: Karpathy, 2015]

Questions to Answer for Yourself / Discuss with Friends

- Repetition: What are the differences between a vanilla RNN unit and an LSTM cell?
- Repetition: Why are vanishing/exploding gradients less of a problem in LSTMs?
- Discuss: Why is it reasonable to say that LSTMs can learn on varying time-scales?

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

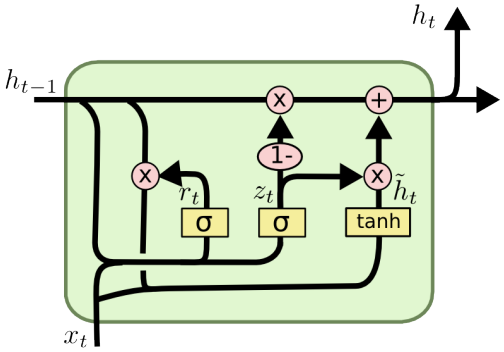
Alternative Models

Frank Hutter Abhinav Valada

University of Freiburg

Gated Recurrent Units (GRUs)

- Gated Recurrent Units (GRUs) are slightly simpler gated architecture compared to LSTMs.
- They also use an input gate r, but only use a single gating unit z to control the forgetting factor and the new contribution to the state update.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

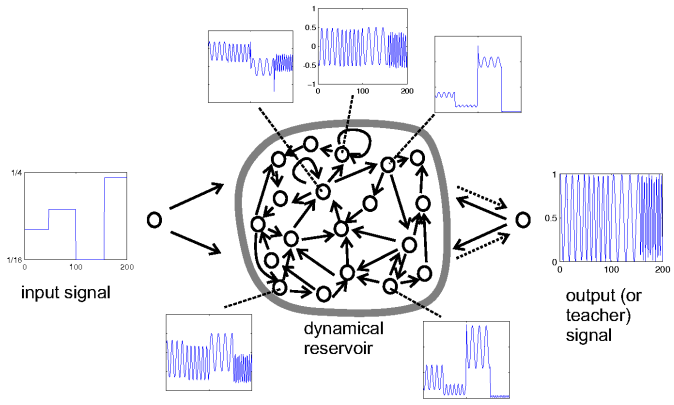
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

[Image source: Olah, 2015 (reproduced with permission)]

Echo State Networks

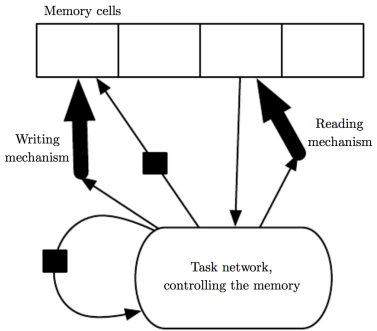
- Idea: use large fixed hidden part (reservoir) with random weights and rich dynamics, scale recurrent weight matrix to spectral radius < 1 (rule of thumb), only train output layer with linear regression.



[Image source: http://www.scholarpedia.org/article/Echo_state_network]

Neural Turing Machines

- Can learn to read from / write to memory cells.
- Training needs gradients $\rightarrow$ use soft addressing with soft-max distribution or hard addressing + reinforcement learning (attention mechanism) .



[Image source: Goodfellow et al., 2016, p. 414, Fig. 10.18]

Lecture Overview

Foundations of Deep Learning, Winter Term 2021/22

Week 7: Recurrent Neural Networks

Summary, Further Reading, References

Frank Hutter Abhinav Valada

University of Freiburg

Summary by Learning Goals

Having heard this lecture, you can now ...

- describe the structure of an RNN.
- describe the computational graphs for RNNs in different sequence processing tasks.
- explain the idea behind BPTT.
- explain the vanishing and exploding gradient problems.
- explain the gating mechanisms of an LSTM and their advantages.
- explain the ideas behind Echo State Networks.

Further Reading

- Nice write-up of the LSTM Forward and Backward pass by Arun Mallya.
- Understanding LSTM Networks by Christopher Olah.
- The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy.

References

Abu-El-Haija, S., Kothari, N., Lee, J., Natsev, P., Toderici, G., Varadarajan, B., Vijayanarasimhan, S. (2016)
YouTube-8M: A Large-Scale Video Classification Benchmark
arXiv preprint arXiv:1609.08675
https://arxiv.org/pdf/1609.08675.pdf

Goodfellow, I., Bengio, Y., Courville, A. (2016)
Deep Learning
MIT Press
https://www.deeplearningbook.org

Hochreiter, S., Schmidhuber, J. (1997)
Long short-term memory
Neural computation 9(8), 1735–1780
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.676.4320&rep=rep1&type=pdf

Karpathy, A. (2015)
The Unreasonable Effectiveness of Recurrent Neural Networks
Andrej Karpathy blog
https://karpathy.github.io/2015/05/21/rnn-effectiveness

Karpathy, A., Fei-Fei, L. (2015)
Deep Visual-Semantic Alignments for Generating Image Descriptions
2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, 2015, pp. 3128–3137, doi: 10.1109/CVPR.2015.7298932
https://arxiv.org/pdf/1412.2306.pdf

Olah, C. (2015)
Understanding LSTM Networks
colah's blog
https://colah.github.io/posts/2015-08-Understanding-LSTMs

Siegelmann, H., Sontag, E. (1991)
Turing Computability With Neural Nets
Applied Mathematics Letters 4, pp. 77–80
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.8383&rep=rep1&type=pdf