Deep Learning
Lecture 5

Thomas Hofmann, ETH Zurich

24 October 2019

Section 9

Convolutional Sequence Models

Models for Sequences

- Many relevant application deal with sequences, e.g. time series (speech, sensors), sequences of symbols (language, biology)
- Vector-valued sequencs: $\mathbf{x}^t \in \mathbb{R}^d$, $t = 0, 1, 2, \ldots$.
- Symbol sequences: $\omega^t \in \Omega$, $t = 0, 1, 2, \ldots$, with Ω: alphabet
  1. variable length sequences, how to represent?
  2. translation invariance, how to incorporate?

From Symbols to Vectors

DNNs operate on real-valued vectors. How can we process (and learn with) symbol sequences? The answer is: via **embeddings**.

Definition (Symbol Embedding)

A d-dimensional symbol embedding is a mapping $z : \Omega \to \mathbb{R}^d$, which maps every symbol to a vector representation.

Definition (Sequence Matrix)

Given a symbol sequence $\omega^1, \ldots, \omega^s$ and an embedding z, the sequence matrix $\mathbf{Z}$ is defined via

$$\mathbf{Z} = \begin{bmatrix} z(\omega^1) & \ldots & z(\omega^s) \end{bmatrix} \in \mathbb{R}^{d \times s}$$

DNN terminology: embedding "layer" = look-up layer.
Shortcut $\mathbf{z}_i := z(\omega_i)$.

Learnable Symbol Representations

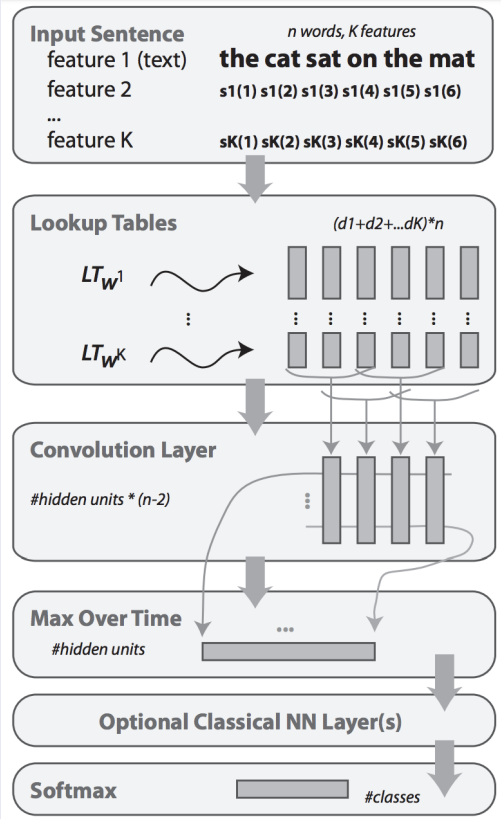
- Embeddings are not fixed, but are learned from data!
- Gradient-based updates via backpropagation, compute $\nabla_{\mathbf{z}_i} \ell$
- Which embeddings are updated depends on which symbols occurred (and how often)
- Learn (just) embeddings in self-supervised learning (a.k.a. representation learning), e.g. word embeddings (word2vec, skipgram, GloVe)

CNNs for Natural Language Processing

Multi-Channel CNNs for NLP

Classical CNN architecture (Collobert & Weston 2008):

1. look-up layers for words and (optionally) linguistic features
2. 1d multi-channel convolution
3. pooling over time (e.g. max)
4. fully connected layers
5. softmax classifier (e.g.)

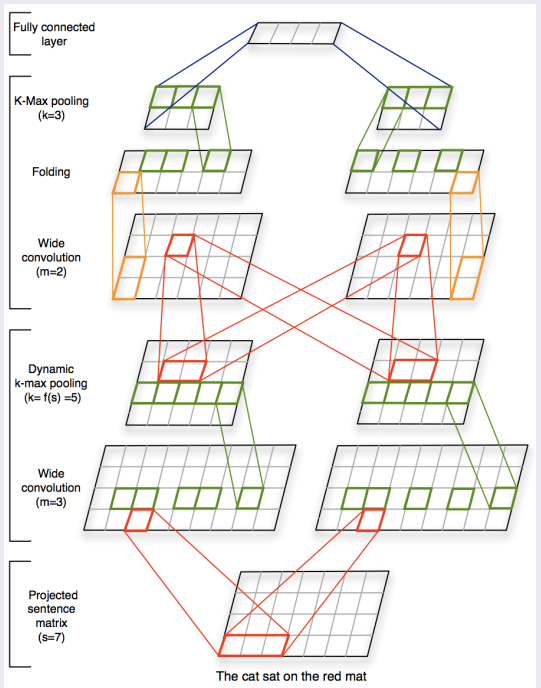


Deep CNNs for Sentence Modeling

Single-Channel Cross-Pooling CNNs for NLP

Alternative modern CNN architecture (Kalchbrenner et al. 2014):

1. embedding layers
2. (wide) one-channel convolutions
3. max-k pooling (order preserving)
4. "dynamic" $= k$ based on data
5. cross-channel interactions via folding (parameter free)



Other work on CNNs for NLP

Themes and variants found in the recent literature

1. Pham et al, 2017, Convolutional neural network language model: avoid max pooling, use batch normalization, (almost) competitve language model
2. Severyn & Moschitti, 2015: Sentiment prediction for tweets

Section 10

Recurrent Networks

State Space Model

State Space Model

Given observation sequence $\mathbf{x}^1, \ldots, \mathbf{x}^s$. Identify hidden activities $\mathbf{h}$ with the state of a dynamical system. Discrete time evolution of **hidden state space sequence**

$$\mathbf{h}^t = F(\mathbf{h}^{t-1}, \mathbf{x}^t; \theta), \quad \mathbf{h}^0 = \mathbf{0}, \quad t = 1, \ldots, s$$

1. Markov property: hidden state at time t depends on input of time t as well as previous hidden state
2. Time-invariance: state evolution function F is independent of time t

Recurrent Neural Network

How should F be chosen?

Recurrent Neural Network

Linear dynamical system w/ elementwise non-linearity

$$\bar{F}(\mathbf{h}, \mathbf{x}; \theta) = \mathbf{W}\mathbf{h} + \mathbf{U}\mathbf{x} + \mathbf{b}, \quad \theta = (\mathbf{U}, \mathbf{W}, \mathbf{b}, \dots)$$

$$F = \sigma \circ \bar{F}, \quad \sigma \in \{\text{logistic, tanh, ReLU, ...}\}$$

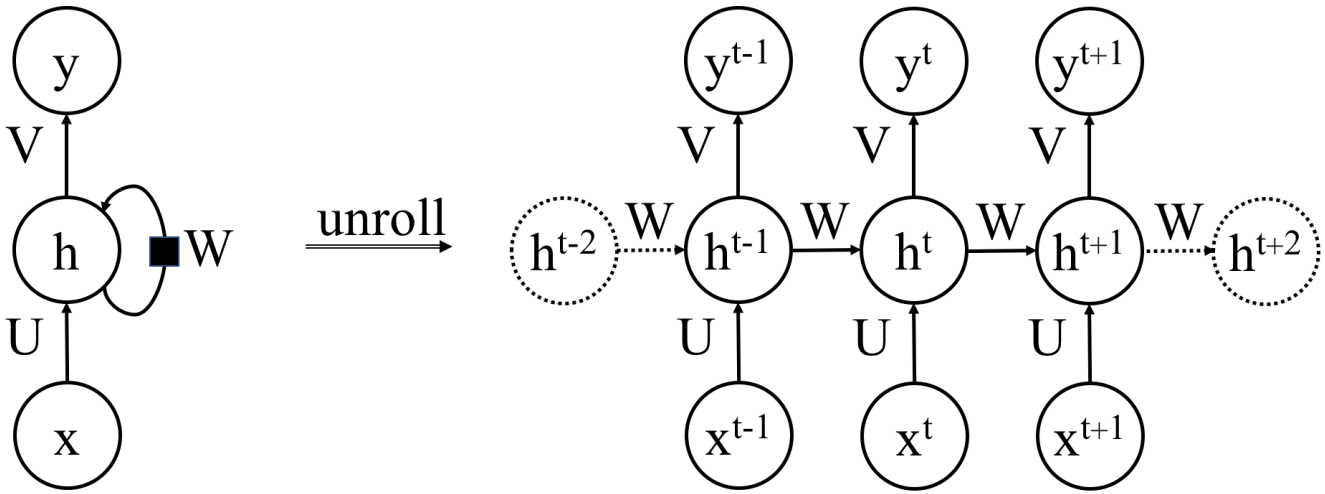
Optionally produce outputs via

$$\mathbf{y} = H(\mathbf{h}; \theta), \quad H(\mathbf{h}; \theta) := \sigma(\mathbf{V}\mathbf{h} + \mathbf{c}), \quad \theta = (\dots, \mathbf{V}, \mathbf{c})$$

Unfolding of Recurrency

Recurrent network: feeding back activities (with time delays).

Unfold computational graph over time (also called unrolling)



Lossy Memorization

What does a recurrent neural network (RNN) do?

1. Hidden state can be thought of as a **noisy memory** or a noisy data summary.
2. Learn to memorize relevant aspects of partial observation sequence:

$$(\mathbf{x}^1, \cdots, \mathbf{x}^{t-1}) \mapsto \mathbf{h}^t$$

3. More powerful than just memorizing fixed-length context.

Feedforward vs. Recurrent Networks

1. For any fixed length s, the unrolled recurrent network corresponds to a feedforward network with s hidden layers.
2. However, inputs are processed in sequence and (optionally) outputs are produced in sequence.
3. Main difference: **sharing of parameters** between layers – same functions F and H at all layers / time steps.

Backpropagation through Time

1. Backpropagation is straightforward: propagate derivatives **backwards through time**.
2. Parameter sharing leads to sum over t, when dealing with derivatives of weights.
3. Define shortcut $\dot{\sigma}_i^t := \sigma'(\bar{F}_i(h^{t-1}, x^t))$, then

$$\frac{\partial \mathcal{R}}{\partial w_{ij}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial w_{ij}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \dot{\sigma}_i^t \cdot h_j^{t-1}$$

$$\frac{\partial \mathcal{R}}{\partial u_{ik}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial u_{ik}} = \sum_{t=1}^{s} \frac{\partial \mathcal{R}}{\partial h_i^t} \cdot \dot{\sigma}_i^t \cdot x_k^t$$

RNN Gradients

RNN where output is produced in last step: $\mathbf{y} = \mathbf{y}^s$

Remember backpropagation in MLPs:

$$\nabla_{\mathbf{x}} \mathcal{R} = \mathbf{J}_{F^1} \cdot \cdots \cdot \mathbf{J}_{F^L} \nabla_{\mathbf{y}} \mathcal{R}$$

Shared weights: $F^t = F$, yet evaluated at different points

$$\nabla_{\mathbf{x}^t} \mathcal{R} = \left[\prod_{r=t+1}^{s} \mathbf{W}^\top \mathbf{S}(\mathbf{h}^r) \right] \cdot \underbrace{\mathbf{J}_H \cdot \nabla_{\mathbf{y}} \mathcal{R}}_{:=\mathbf{z}}$$

where

$$\mathbf{S}(\mathbf{h}^r) = \text{diag}(\dot{\sigma}_1^r, \ldots, \dot{\sigma}_n^r)$$

which is $\leq \mathbf{I}$ for $\sigma \in \{\text{logistic, tanh, ReLU}\}$

Exploding and/or Vanishing Gradients

Spectral norm of matrix: largest singular value

$$\|\mathbf{A}\|_2 = \max_{\mathbf{x}:\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \sigma_{\max}(\mathbf{A})\,.$$

Note that $\|\mathbf{AB}\|_2 \leq \|\mathbf{A}\|_2 \cdot \|\mathbf{B}\|_2$, hence with $\mathbf{S}(\cdot) \leq \mathbf{I}$

$$\left\| \prod_{s=t+1}^{s} \mathbf{W}^\top \mathbf{S}(\mathbf{h}^t) \right\|_2 \leq \left\| \prod_{s=t+1}^{s} \mathbf{W}^\top \right\|_2 \leq \|\mathbf{W}\|_2^{s-t} = [\sigma_{\max}(\mathbf{W})]^{s-t}$$

If $\sigma_{\max}(\mathbf{W}) < 1$, gradients are vanishing, i.e.

$$\|\nabla_{\mathbf{x}^t} \mathcal{R}\| \leq \sigma_{\max}(\mathbf{W})^{s-t} \cdot \|\mathbf{z}\| \overset{(s-t)\to\infty}{\longrightarrow} 0$$

Conversely, if $\sigma_{\max}(\mathbf{J}_F) > 1$ gradients may explode.

(Depends on gradient direction, Pascanu, Mikolov, Bengio, 2013)

Bi-directional Recurrent Networks

Hidden state evolution does not always have to follow direction of time (or causal direction).

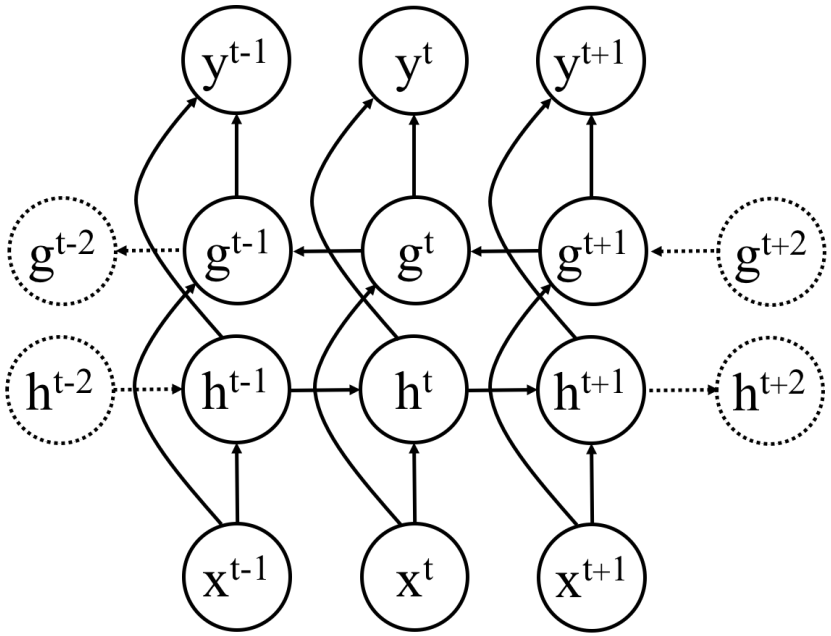
Define reverse order sequence

$$\mathbf{g}^t = G(\mathbf{x}^t, \mathbf{g}^{t+1}; \theta)$$

as model w/ separate parameters.

Now we can interweave hidden state sequences (see figure).

Backpropagation is also bi-directional.



Deep Recurrent Networks

Deep recurrent networks:

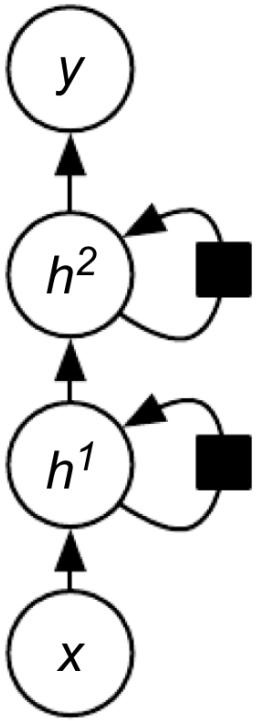
hierarchical hidden state:

$$\mathbf{h}^{t,1} = F^1(\mathbf{h}^{t-1,1}, \mathbf{x}^t; \theta),$$
$$\mathbf{h}^{t,l} = F^l(\mathbf{h}^{t-1,l}, \mathbf{h}^{t,l-1}; \theta) \quad (l = 1, \ldots, L)$$

Output connected to last hidden layer

$$\mathbf{y}^t = H(\mathbf{h}^{t,L}; \theta)$$

Can be combined with bi-directionality (how?).



Section 11

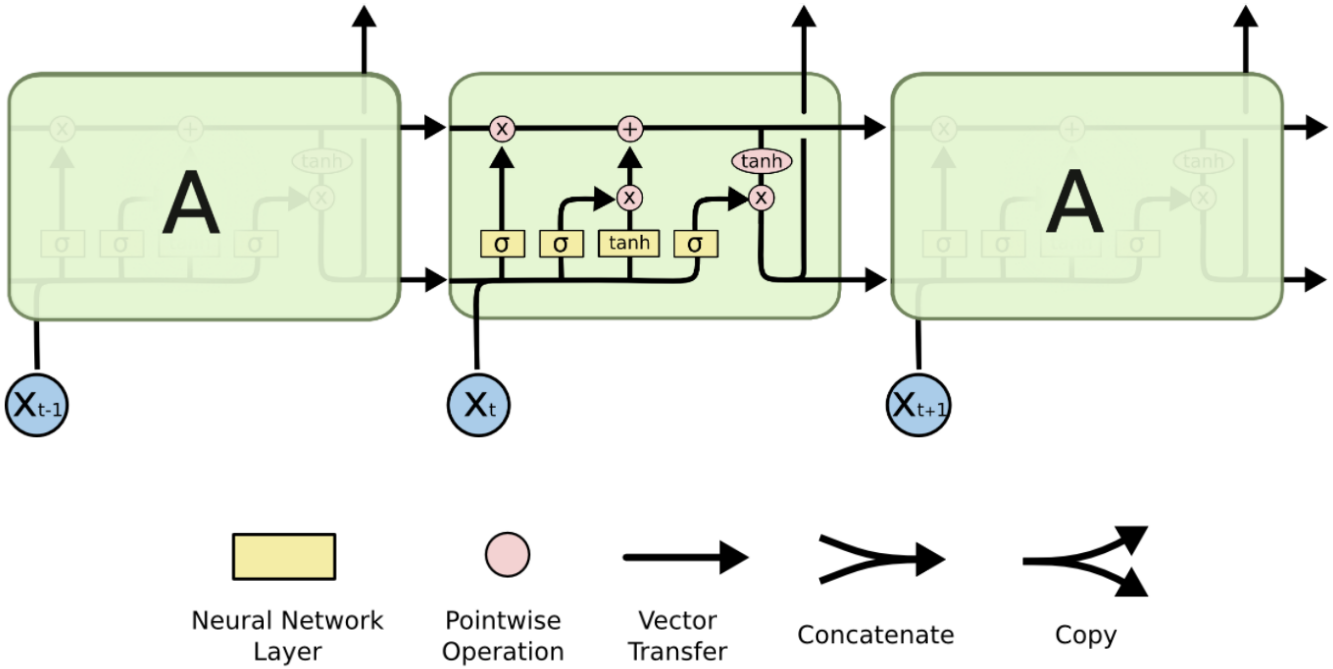
Memory Units

Long-Term Dependencies

1. Sometimes: important to model long-term dependencies $\Longrightarrow$ network needs to memorize features from the distant past
2. Recurrent network: hidden state needs to preserve memory
3. Conflicts with short-term fluctuations and vanishing gradients
4. Conclusion: difficult to learn long-term dependencies with standard recurrent network (see DL Section 10.7)
5. Popular remedy: **gated units**

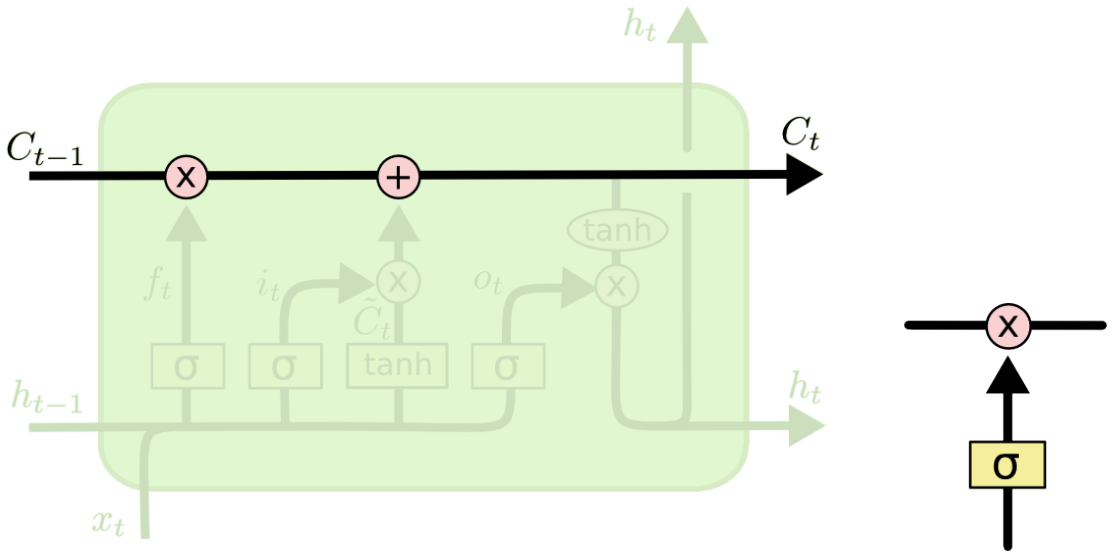
LSTM: Overall Architecture

Long-Short-Term-Memory: unit for memory management



from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

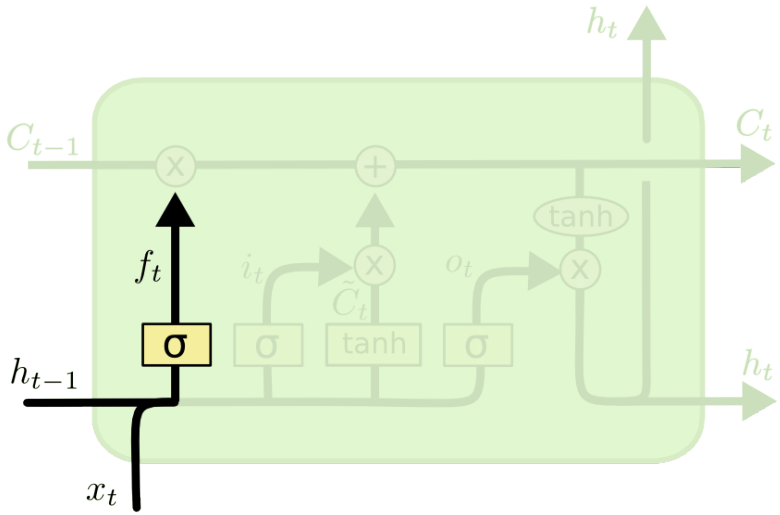
LSTM: Flow of Information



1. Information propagates along the chain like on a conveyor belt.
2. Information can flow unchanged and is only selectively changed (vector addition) by σ-gates.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Forget Gate

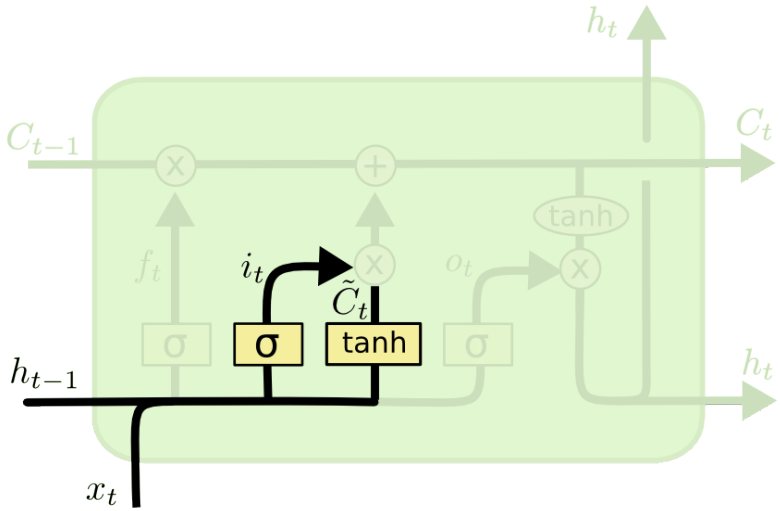


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

1. Keeping or forgetting of stored content?

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Input $\rightarrow$ Memory Value



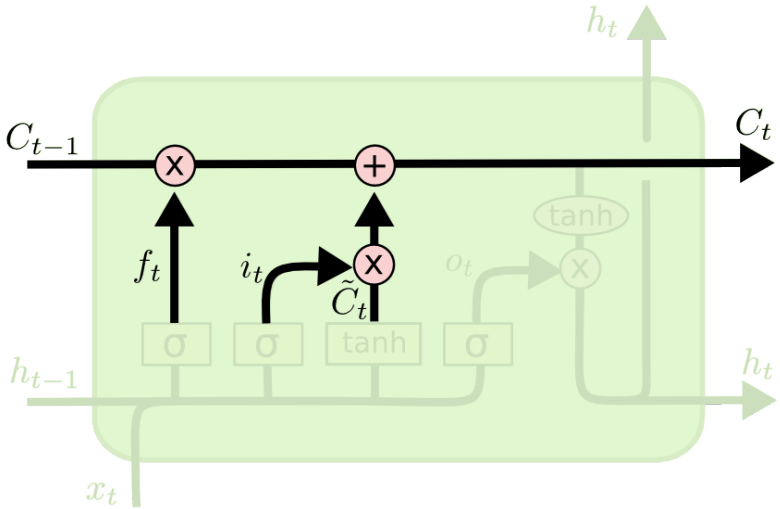
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

1. Preparing new input information to be added to the memory.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Updating Memory

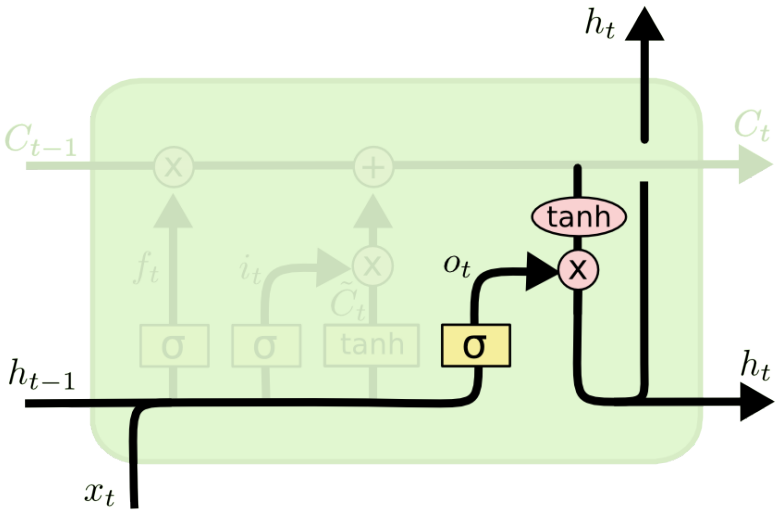


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

1. Combining stored and new information.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

LSTM: Output Gate



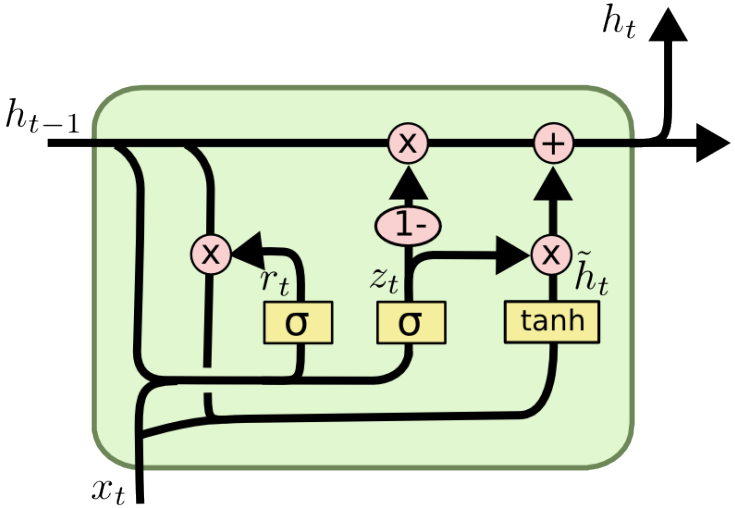
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1. Computing output selectively.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Gated Memory Units



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

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

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

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

1. Memory state = output. Modifications to logic (Cho et al, 2014).
2. Convex combination of old and new information.

from Christopher Olah: `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

Gated Memory Units

GRUs and LSTMs can learn active memory strategies: what to memorize, overwrite and recall when.

Successful use cases

- handwriting recognition
- speech recognition (also: Google)
- machine translation
- image captioning

Notoriously difficult to understand what units learn...

Resource-hungry. Slow in learning.

Language Modeling

Model	Test Perplexity	Number of Params [billions]
Sigmoid-RNN-2048 (Ji et al., 2015a)	68.3	4.1
Interpolated KN 5-gram, 1.1B n-grams (Chelba et al., 2013)	67.6	1.76
Sparse Non-Negative Matrix LM (Shazeer et al., 2015)	52.9	33
RNN-1024 + MaxEnt 9-gram features (Chelba et al., 2013)	51.3	20
LSTM-512-512	54.1	0.82
LSTM-1024-512	48.2	0.82
LSTM-2048-512	43.7	0.83
LSTM-8192-2048 (No Dropout)	37.9	3.3
LSTM-8192-2048 (50% Dropout)	32.2	3.3
2-Layer LSTM-8192-1024 (BIG LSTM)	30.6	1.8
BIG LSTM+CNN Inputs	**30.0**	**1.04**
BIG LSTM+CNN Inputs + CNN Softmax	39.8	**0.29**
BIG LSTM+CNN Inputs + CNN Softmax + 128-dim correction	35.8	**0.39**
BIG LSTM+CNN Inputs + Char LSTM predictions	47.9	**0.23**

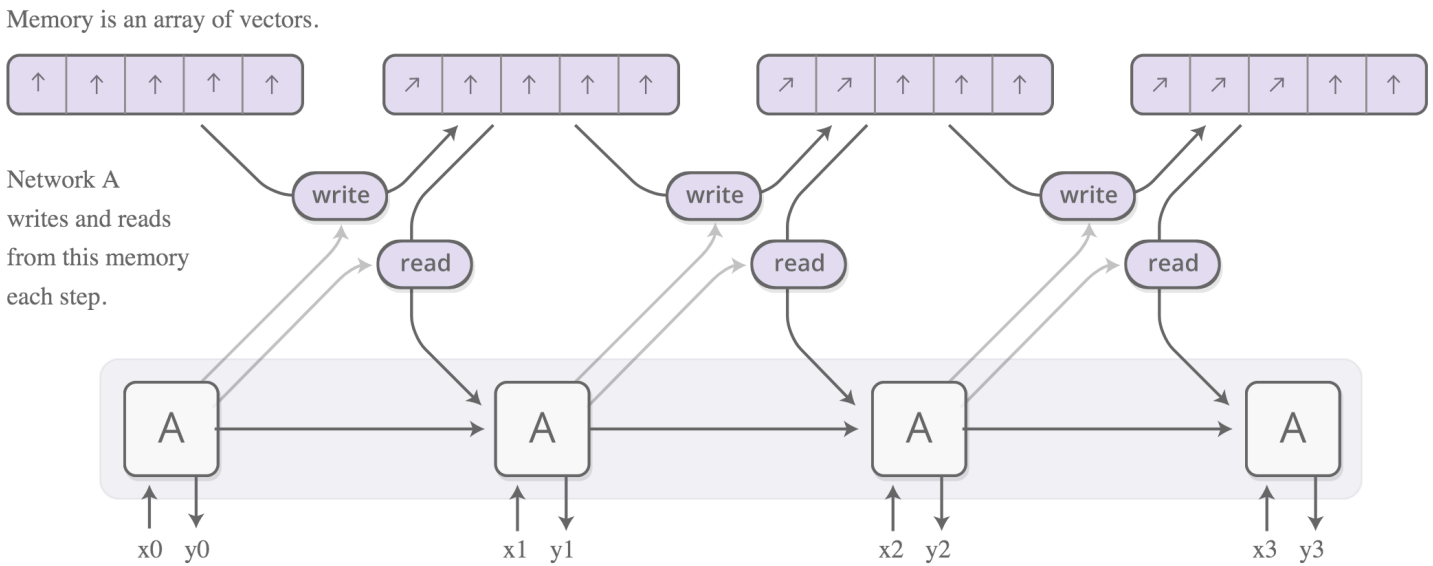
from Jozefowicz et al, 2016

1. evaluation on corpus w/ 1B words
2. number of parameters can be in the 100Ms or even Bs!
3. ensembles can reduce perplexity to ≈ 23 (best result 06/2016)

Section 12

Differentiable Memory

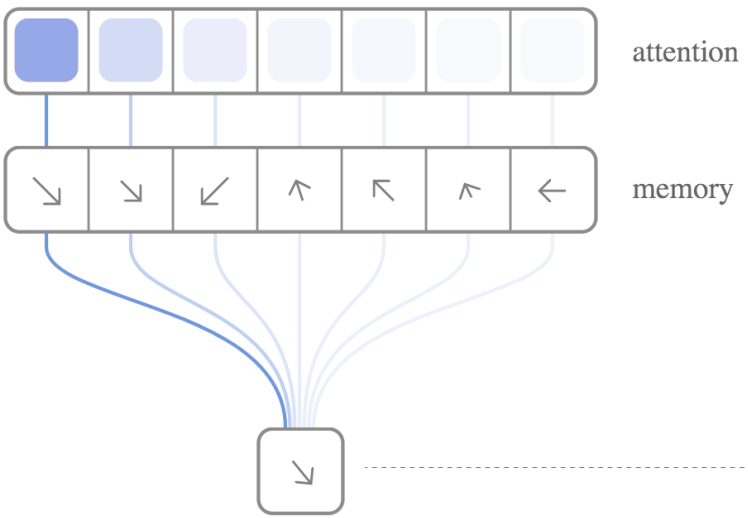
Neural Turing Machine: Architecture

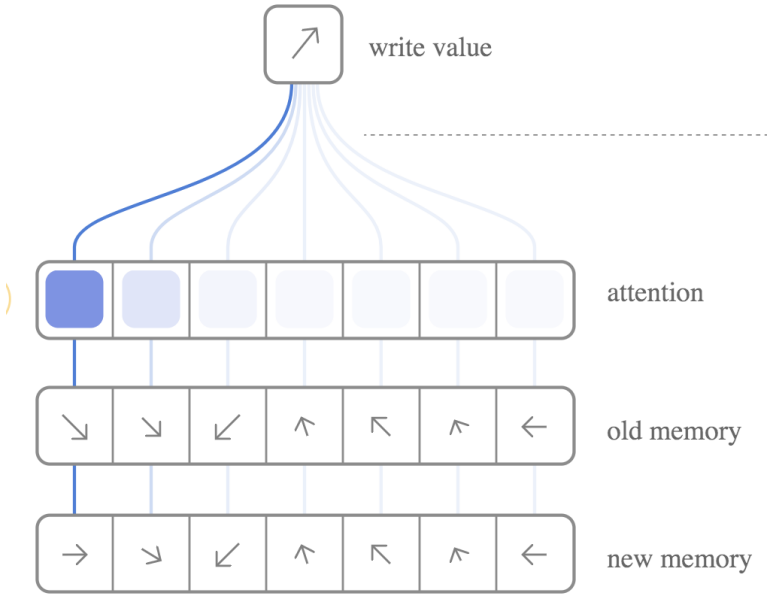


1. RNN controls an external memory bank
2. Reminiscent of Turing machine, but: each cell $M_i \in \mathbb{R}^d$

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Neural Turing Machine: Differentiable Memory

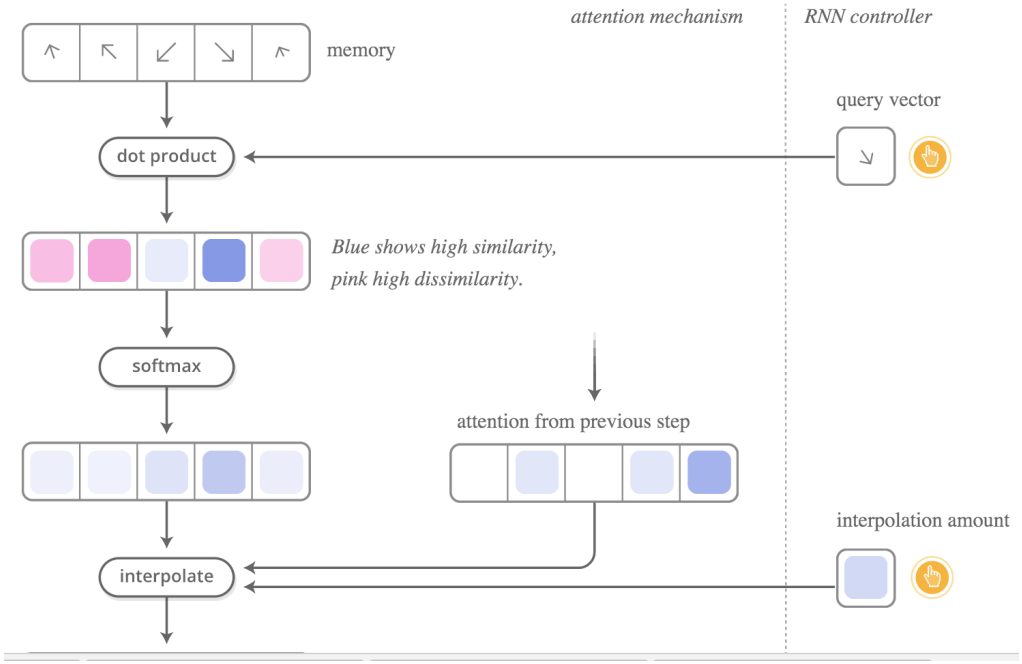




1. Compute attention distribution $(\alpha_i)_i$, $\alpha_i \geq 0$ s.t. $\sum_i \alpha_i = 1$.
2. Read out expected memory content: $r \leftarrow \sum_i \alpha_i M_i$.
3. Write uses weights $(\beta_i)_i$, $\beta_i \in [0; 1]$, $M_i \leftarrow (1 - \beta_i) M_i + \beta_i w$.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

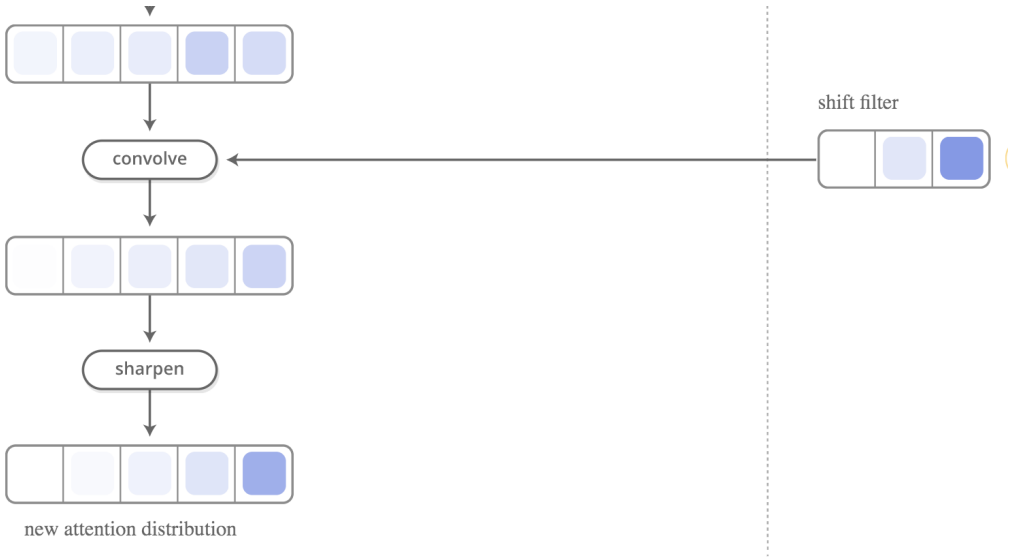
Neural Turing Machine: Memory Controller



1. Associative memory access with query (key) $q \in \mathbb{R}^d$.
2. Cells are scored via softmax.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Neural Turing Machine: Memory Controller



1. Ability to shift relative to content-selected locations (convolution).
2. Additional accentuation to sharpen attention distribution.

from Olah & Carter, 2016: `http://distill.pub/2016/augmented-rnns`

Differentiable Memory: Discussion

1. NTM architectures can learn loops and simple programs. However, no real-world applications.
2. Other architectures:
   - Neural random access machines (Kurach et al, 2015)
   - Differentiable data structures like stacks and queues (Grefenstette et al., 2015; Joulin & Mikolov, 2015)
3. Direction of future research