Deep Learning

Lecture 8

Fernando Perez-Cruz
based on Thomas Hofmann lectures

Swiss Data Science Center
ETH Zurich and EPFL – `datascience.ch`

November 12, 2018

Overview

1. Convolutional Networks for Natural Language

2. Recurrent Networks

Section 1

Convolutional Networks for Natural Language

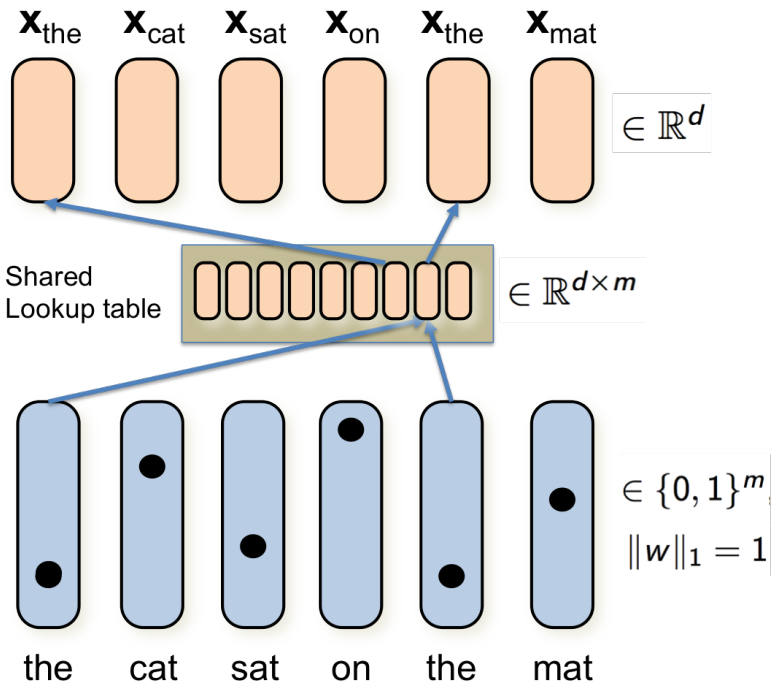
Symbolic Units

Fundamental Problem: language is composed of symbols
(e.g. characters, words)

- no "signals" or continuous measurements (vs. vision, speech)
- connection between phonemes/morphemes and meaning largely based on convention
  - i.e. symbols can be re-identified, but are otherwise arbitrary
- conventions are expressed in repeated usage of words in context

Word Embeddings

Basic idea: map symbols over vocabulary $\mathcal{V}$ to vector representation = embedding into a (Euclidian) vector space

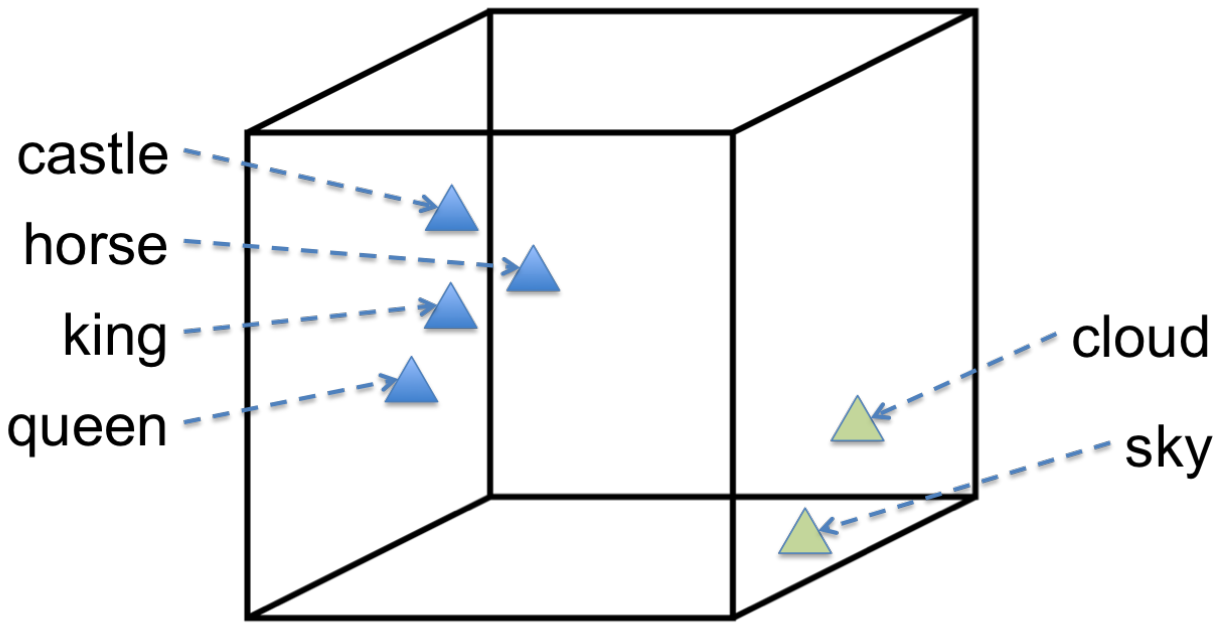


$$\text{symbolic} \quad \mathcal{V} \ni w \overset{\text{embed}}{\mapsto} \mathbf{x}_w \in \mathbb{R}^d \quad \text{quantitative}$$

Word Vector Model: Illustration

- Vector representation of words = embedding

$$w_i \mapsto \mathbf{x}_i \in \mathbb{R}^D$$



Bi-linear Models

Interpret vectors as latent variables and link them to observable through a probabilistic model.

Simplest case: pairwise word co-occurrence $\Longrightarrow$ bilinear model

$$\text{pmi}(v, w) = \log \frac{p(v, w)}{p(v)p(w)} = \log \frac{p(v|w)}{p(v)} = \mathbf{x}_v^\top \mathbf{x}_w + \text{const}$$

- pointwise mutual information related to inner product between latent vectors

Skip-gram: Setting

Pairwise occurrences of words in context window (of size R)

Simplified: text = long sequence of words, $\mathbf{w} = (w_1, \ldots, w_T)$.

Co-occurrence index set

$$\mathcal{C}_R := \{(i, j) \in [1 : T]^2 : 1 \leq |i - j| \leq R\}.$$

Skip-gram: Setting

Pairwise occurrences of words in context window (of size R)

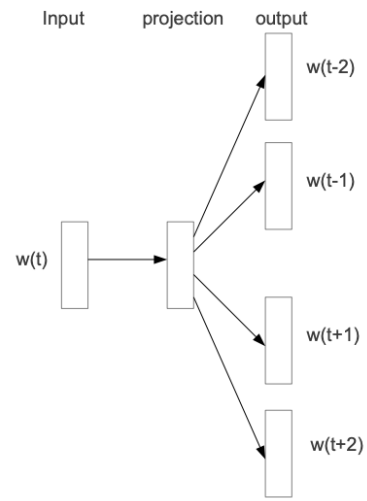
Simplified: text = long sequence of words, $\mathbf{w} = (w_1, \ldots, w_T)$.

Co-occurrence index set

$$\mathcal{C}_R := \{(i, j) \in [1 : T]^2 : 1 \leq |i - j| \leq R\}.$$

Skip-gram objective (Mikolov et al, 2013):

$$\mathcal{L}(\theta; \mathbf{w}) = \sum_t \sum_{\substack{\ell=-c \\ \ell \neq 0}}^{c} \log p_\theta(w_{t+\ell} | w_t)$$



Skip-gram: Model

Skip-gram model (soft-max)

$$\log p_\theta(v|w) = \mathbf{x}_v^\top \mathbf{z}_w - \log \underbrace{\sum_{u \in \mathcal{V}} \exp\left[\mathbf{x}_u^\top \mathbf{z}_w\right]}_{\text{partition function}}$$

- two vectors per word (allow for asymmetry), $\theta = (\mathbf{x}_w, \mathbf{z}_w)_{w \in \mathcal{V}}$
- expensive to compute log-partition function
  - sum over $\mathcal{V}$ (can be $\sim 10^5 - 10^7$)
  - how can skip-gram be made more efficient?

Skip-Gram: Negative Sampling

Negative sampling distribution

$$p_n(w) = p(w)^\alpha, \;\; p(w) = \text{relative word frequency},$$

- typically $\alpha \approx \frac{3}{4}$
- use this to generate "random word pairs"

Re-formulate as logistic regression

$$\mathcal{L}(\theta; \mathbf{w}) = \sum_{(i,j) \in C_R} \Big[\underbrace{\log \sigma(\mathbf{x}_{w_i}^\top \mathbf{z}_{w_j})}_{\text{positive examples}} + k \; \mathbf{E}_{v \sim p_n} [\underbrace{\log \sigma(-\mathbf{x}_{w_i}^\top \mathbf{z}_v)}_{\text{negative examples}}] \Big]$$

- $k \approx 2 - 10$ oversampling factor

Skip-Gram Model

Results: semantic analogies captured by word embeddings

Newspapers			
New York	New York Times	Baltimore	Baltimore Sun
San Jose	San Jose Mercury News	Cincinnati	Cincinnati Enquirer
NHL Teams			
Boston	Boston Bruins	Montreal	Montreal Canadiens
Phoenix	Phoenix Coyotes	Nashville	Nashville Predators
NBA Teams			
Detroit	Detroit Pistons	Toronto	Toronto Raptors
Oakland	Golden State Warriors	Memphis	Memphis Grizzlies
Airlines			
Austria	Austrian Airlines	Spain	Spainair
Belgium	Brussels Airlines	Greece	Aegean Airlines
Company executives			
Steve Ballmer	Microsoft	Larry Page	Google
Samuel J. Palmisano	IBM	Werner Vogels	Amazon

From Words to Sequences of Words

Question: Can we extend word embeddings to embeddings for sequences of words?

Why is this relevant? Fundamental question of statistical language modeling (cf. Shannon).

$$\text{estimate } p(w_1 \dots w_T) = \prod_{t=1}^{T} p(w_t | w_{t-1} \dots w_1)$$

- predict next word in a sequence of words
- quality measured by perplexity (exp of average log-probability)

From Words to Sequences of Words

Traditional approach: k-th order Markov assumption

$$p(w_t|w_{t-1}\dots w_1) \approx p(w_t|w_{t-1}\dots w_{t-k}) \approx (k+1)\text{-gram counts}$$

- in practice: 5-grams, i.e. $k = 4$
- advanced smoothing techniques (Kneser-Ney smoothing)

Modern approach: language models via embeddings

$$\log p(v|\mathbf{w} := w_1, \dots w_{t-1}) = \mathbf{x}_v^\top \mathbf{z}_\mathbf{w} + \text{const}$$

- $\mathbf{x}_v$ = word embedding
- $\mathbf{z}_\mathbf{w}$ = sequence embedding
- Question: how can we construct sequence embeddings?

From Words to Sequences of Words

Three main approaches:

1. Convolutional networks
   - pros: conceptually simple, fast to train
   - cons: limited range memory

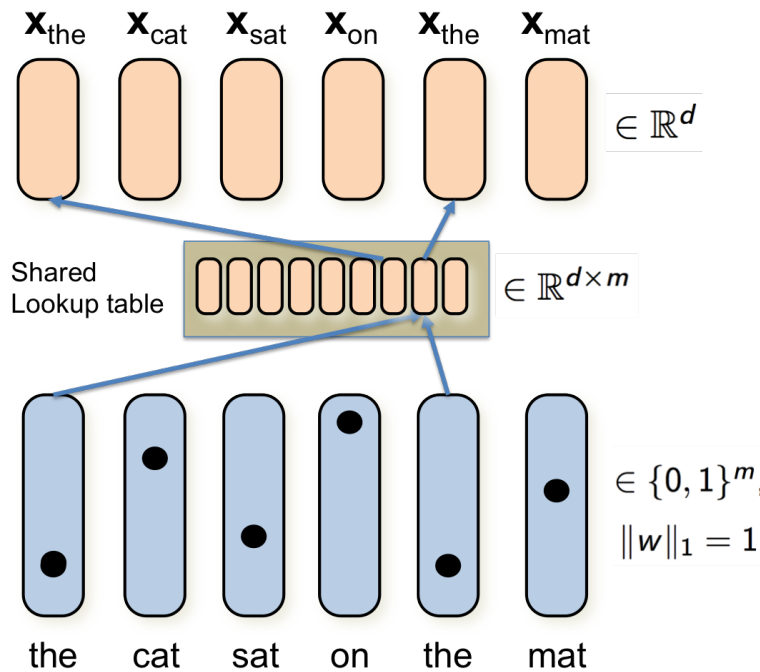
2. Recurrent networks
   - pros: active memory management via gated units
   - cons: more difficult to optimize, larger data sets needed

3. Recursive networks (in combination with parsers)

ConvNets: Word Representations

Idea: use convolutional network on top of word embeddings – take a deep learning view!

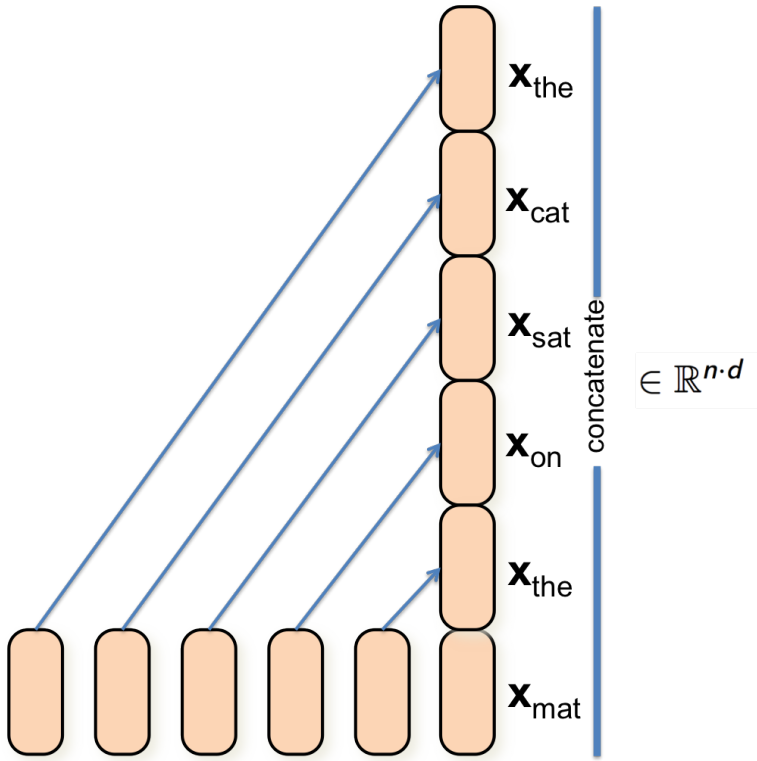
Step 1: map word sequence to vector sequence = sentence matrix



ConvNets: Concatenation

Step 2: concatenate vectors

(variable length)



ConvNets: Channels

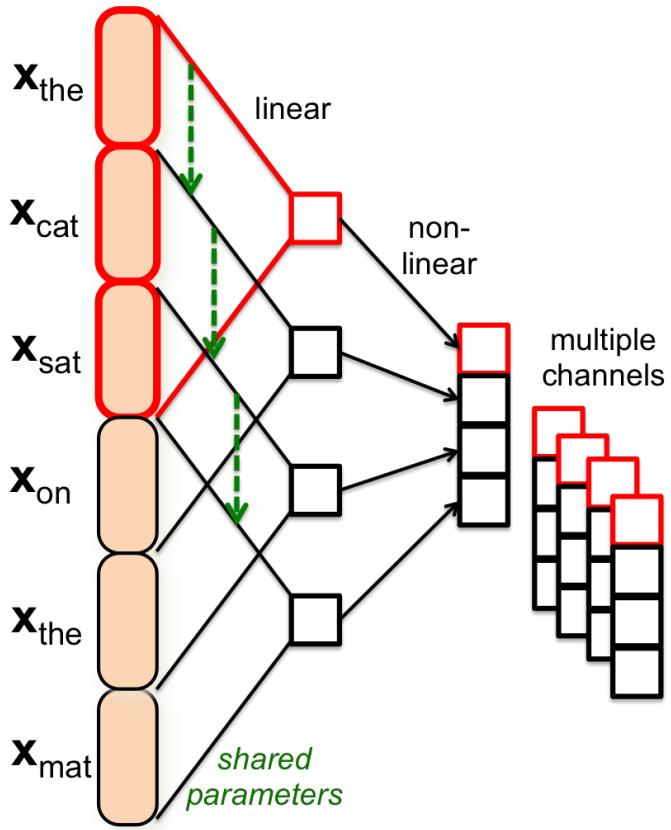
Step 3: convolve

chose window size(s), e.g. 3 or 5

single channel $f_j : \mathbb{R}^{3d} \rightarrow \mathbb{R}$

- shift mask over input
- parameter sharing (filters)
- nonlinearity (tanh or ReLU)

k independent channels

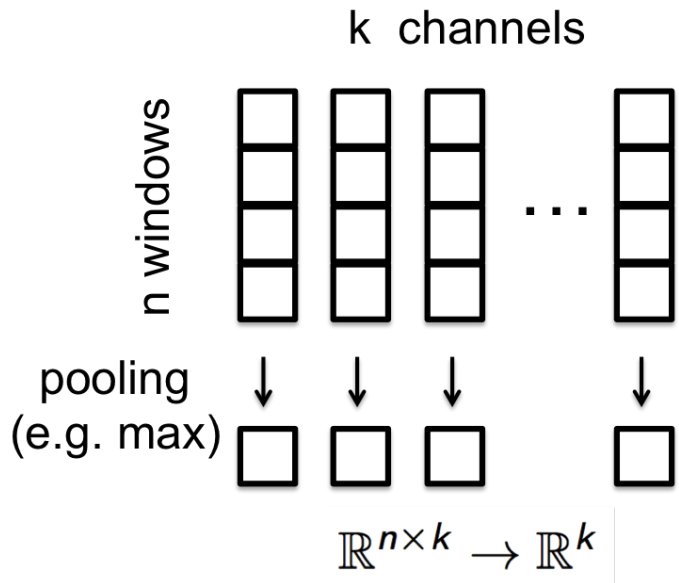


ConvNets: Pooling

Step 4: pooling

reduce each channel (variable length) to single number

- mathematically: $\mathbb{R}^{n \times k} \rightarrow \mathbb{R}^k$
- typically: max-over-time pooling (no parameters)
- map word sequence to fixed-length representation



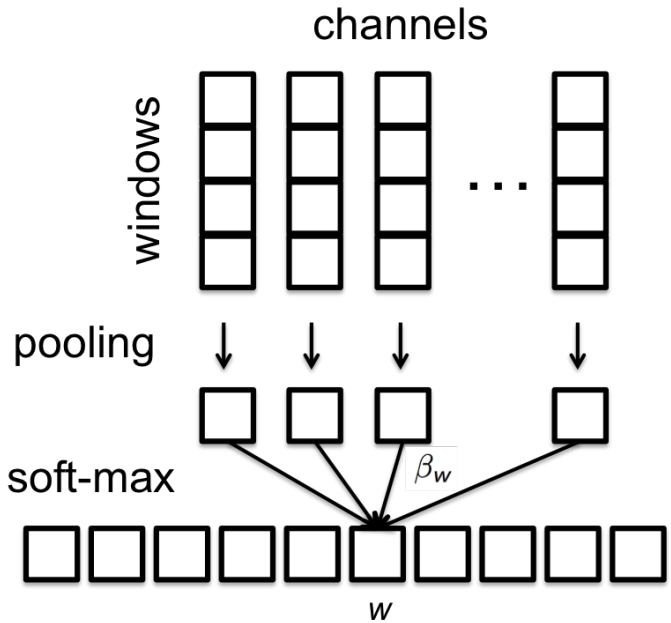
ConvNets: Softmax

Step 5: SoftMax

predict words via soft-max

$$p(v|\mathbf{w}) = \frac{\exp\left[\langle \mathbf{y}_v, \mathbf{z}_\mathbf{w} \rangle\right]}{\sum_{u \in \mathcal{V}} \exp\left[\langle \mathbf{y}_u, \mathbf{z}_\mathbf{w} \rangle\right]}$$

- uses word embeddings $\mathbf{y}_v \in \mathbb{R}^k$
- two types of word embeddings ($\mathbf{x}_v \in \mathbb{R}^d$, $\mathbf{y}_v \in \mathbb{R}^k$)



Sentence ConvNets: Variants

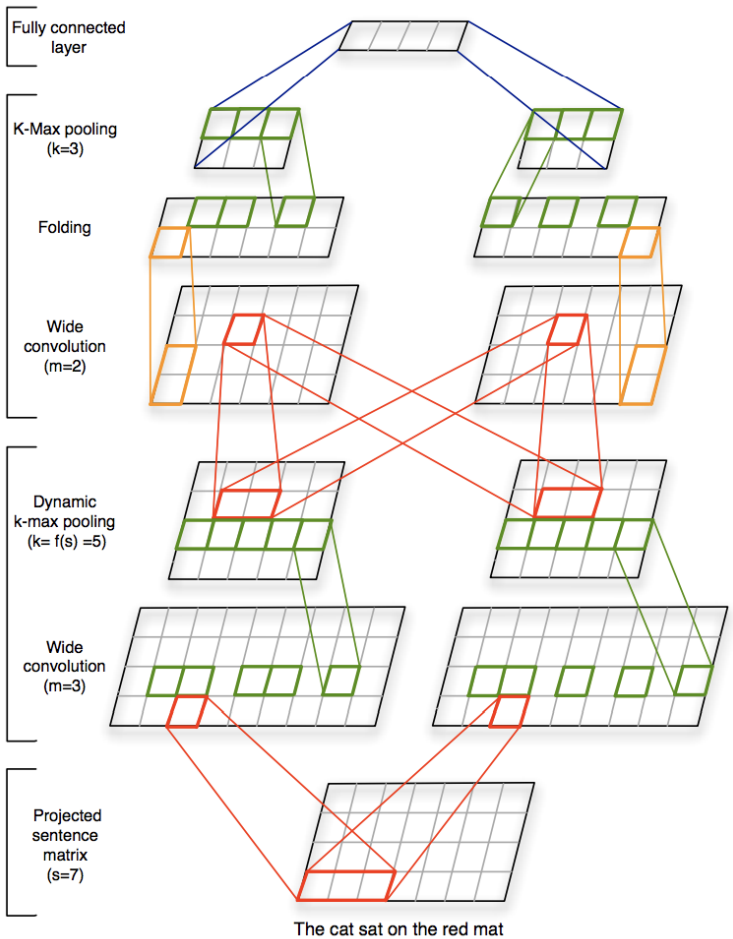
The above architecture follows Collobert & Weston, 2008, Collobert et al., 2011.

- it is also called time-delay NN with max-over-time pooling

Variants suggested in Kalchbrunner, Greffenstette & Blunsom, 2014:

- wide convolutions w/ zero padding
  - better coverage of beginning/end of sentence
- dynamic max-k pooling
  - stack convolutions, pooling window size decreasing with layers
- folding to combine information across embedding dimensions

Dynamic CNNs



from Kalchbrunner et al., 2014

ConvNets: Applications

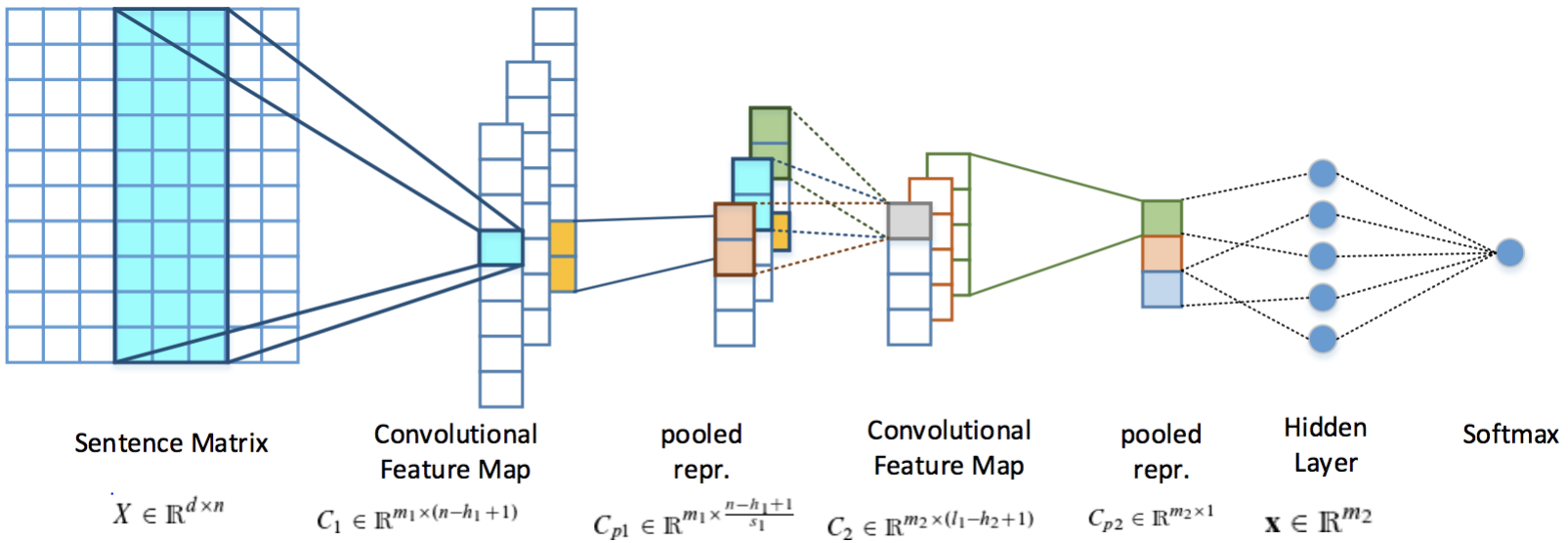
ConvNet architecture was used in early neural language models. (Bengio et al, 2003; Bengio et al, 2006)

Can be extended to character level models (no word segmentation).

Useful architecture for supervised tasks – example:

Sentiment prediction for tweets (cf. Severyn & Moschitti, 2015, Deriu et al., 2017)

ConvNets: Sentiment Classification



from Deriu et al., 2017

Section 2

Recurrent Networks

Discrete Time Systems

Given observation sequence $\mathbf{x}^1, \ldots, \mathbf{x}^T$.

Identify hidden activities $\mathbf{h}$ with the state of a dynamical system.

Discrete time evolution of hidden state sequence

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

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

Generalized Linear Dynamical System

How should F be chosen?

Linear dynamical system:

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

As always: compose with elementwise non-linearity

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

Optionally produce outputs via

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

Sequence Loss Functions

Two typical settings:

1. output at the end ($t = T$: last step): $\mathbf{h}^T \mapsto H(\mathbf{h}^T; \theta) = \mathbf{y}$

Same as feedforward network: use loss J on $\mathbf{y}$.

2. output at every time step: sequence $\mathbf{y}^1, \ldots, \mathbf{y}^T$

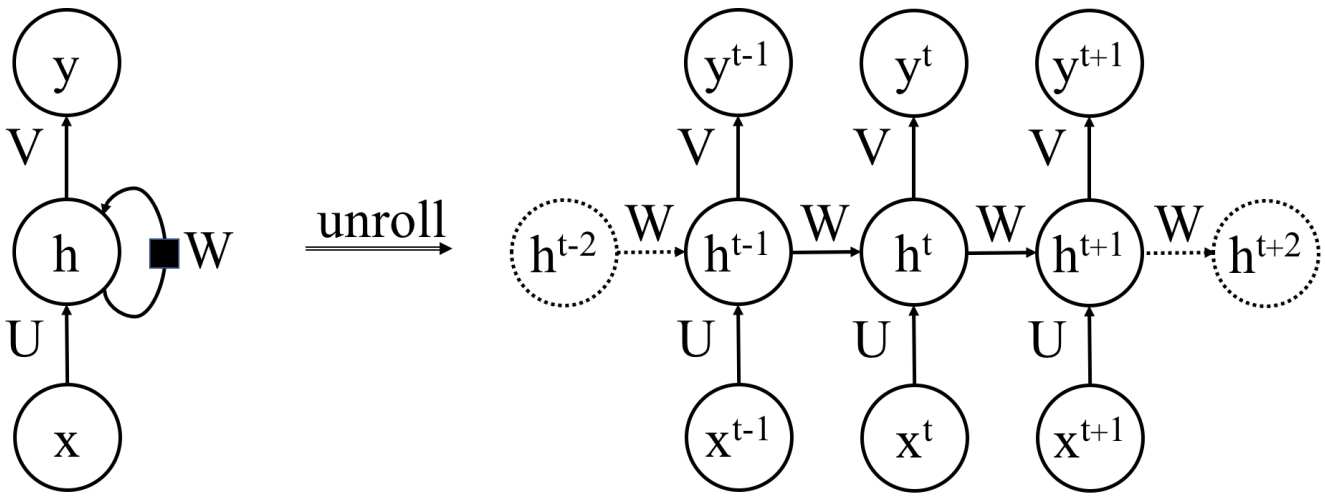
Additive loss function

$$\mathcal{R}(\mathbf{y}^1, \ldots \mathbf{y}^T) = \sum_{t=1}^{T} \mathcal{R}(\mathbf{y}^t) = \sum_{t=1}^{T} \mathcal{R}(H(\mathbf{h}^t; \theta))$$

Unfolding

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?

Hidden state can be thought of as a noisy memory or a noisy data summary.

Learn to memorize relevant aspects of partial observation sequence:

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

More powerful than just memorizing fixed-length context.

Feedforward vs. Recurrent Networks

For any fixed T, the unrolled recurrent network corresponds to a feedforward network with T hidden layers.

However, inputs are processed in sequence and (optionally) outputs are produced in sequence.

Main difference: sharing of parameters between layers – same functions F and H at all layers / time steps.

Backpropagation in Recurrent Networks

Backpropagation is straightforward: propagate derivatives backwards through time.

Parameter sharing leads to sum over t, when dealing with derivatives of weights.

Define shortcut $\dot{\sigma}_i^t := \sigma'(\bar{F}_i(h^{t-1}, x^t))$, then

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

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

Exploding and/or Vanishing Gradients

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

Remember backpropagation in MLPs:

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

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

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

where

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

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

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

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

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

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).

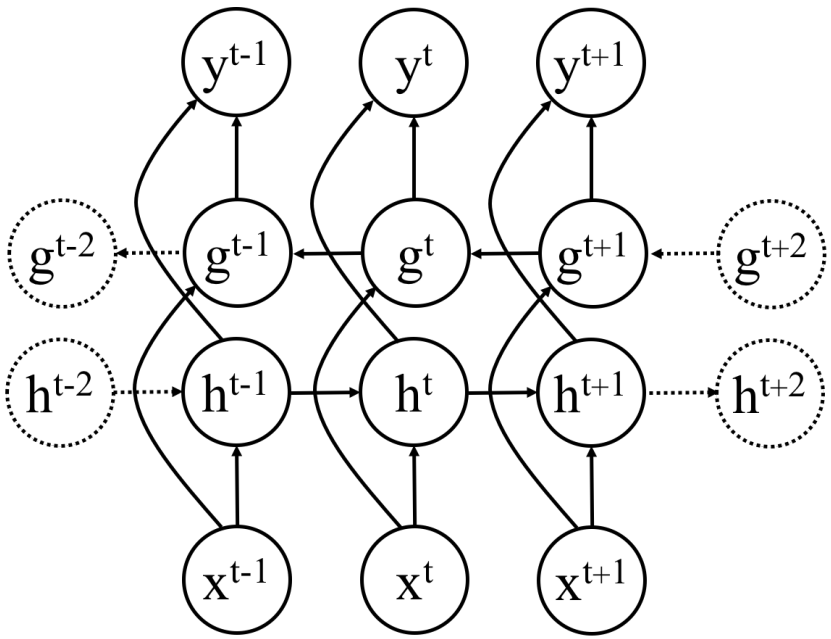
Define reverse order sequence

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

as model w/ separate parameters.

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

Backpropagation is also bi-directional.



Deep Recurrent Networks

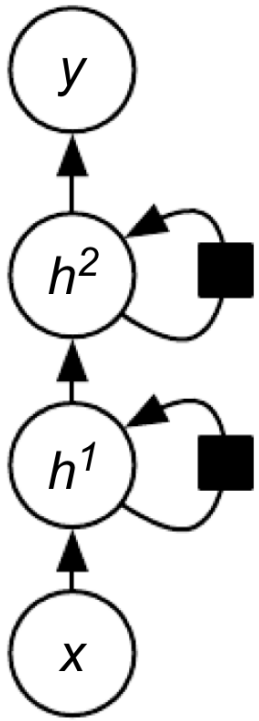
Deep recurrent networks:

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.

Probability Distribution over Sequences

Goal: define conditional probability distribution over output sequence $\mathbf{y}^{1:T}$, given input sequence $\mathbf{x}^{1:T}$.

Step-by-step prediction:

$$p(\mathbf{y}^{1:T}|\mathbf{x}^{1:T}) \approx \prod_{t=1}^{T} p(\mathbf{y}^t|\mathbf{x}^{1:t}, \mathbf{y}^{1:t-1})$$

Naive implementation by RNN

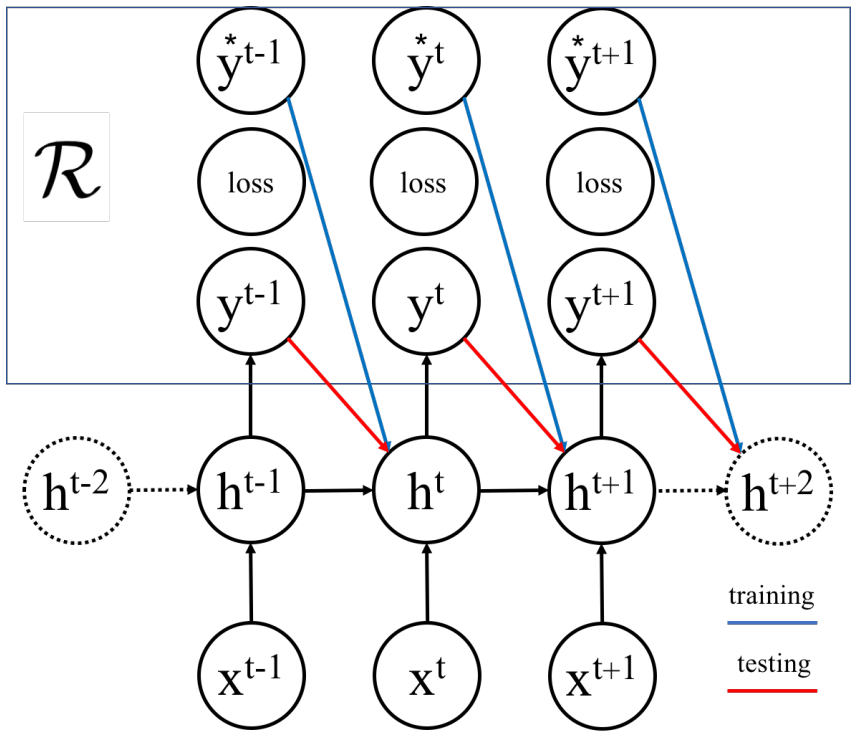
$$\mathbf{x}^{1:t} \overset{F}{\mapsto} \mathbf{h}^t, \quad \mathbf{h}^t \overset{H}{\mapsto} \mu^t \quad \mu^t \mapsto p(\mathbf{y}^t)$$

Problem: $p(\mathbf{y}^t)$ depends on $\mathbf{y}^{1:t-1}$ only through $\mathbf{h}^t$.

(example: phrases - e.g. "New York" or "Los Angeles" but not "New Angeles" or "Los York")

Output Feedback

How can we overcome this? Feeding-back of previous outputs.



During training: teacher forcing

During prediction, feeding-back generated outputs

Bibliography

- F. Chaubard, R. Mundra, R. Socher, (2017). "Lectures Note 1: CS 224D: Deep Learning for NLP". http://cs224d.stanford.edu/lecture_notes/notes1.pdf
- R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuogl and P. Kuksa, (2011). "Natural language processing (almost) from scratch," JMLR 12
- I. Goodfellow, Y. Bengio and A. Courville, (2016). "Deep Learning". MIT Press (Chapter 10).