TTIC 31210:
Advanced Natural Language Processing

Kevin Gimpel

Spring 2017

Lecture 3:
Word Embeddings

Assignment 1

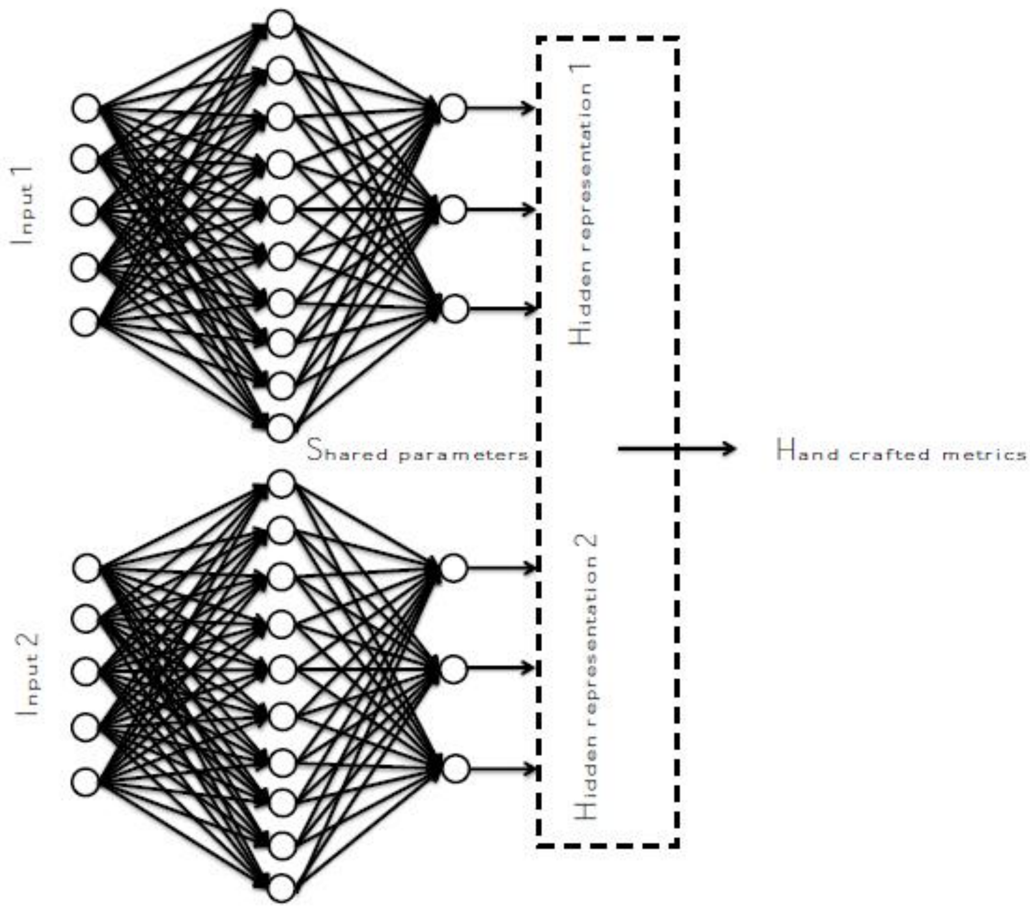
- Assignment 1 due tonight

Roadmap

- review of TTIC 31190 (week 1)
- **deep learning for NLP (weeks 2-4)**
- generative models & Bayesian inference (week 5)
- Bayesian nonparametrics in NLP (week 6)
- EM for unsupervised NLP (week 7)
- syntax/semantics and structure prediction (weeks 8-9)
- applications (week 10)

Neural Similarity Modeling

- “Siamese networks” (Bromley et al., 1993)
  - two identical networks with shared parameters
  - at end, similarity computed between two representations



Similarity Functions

- many choices for similarity functions
- we talked about some during Lecture 2

Learning for Similarity

- We want to learn input representation function $f_{\boldsymbol{\theta}}$ as well as any parameters of similarity function
- We'll just write all these parameters as $\boldsymbol{\theta}$
- How about this loss? (loss A on your handout)

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} -sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2))$$

- Any potential problems with this?

(Better) Learning for Similarity

- Contrastive hinge loss (loss B on handout):

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} [-sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)) + sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{v}))]_+$$

$$[a]_+ = \max(0, a)$$

- $\boldsymbol{v}$ is a "negative" example
- Any potential problems with this?

(Better) Learning for Similarity

- Large-margin contrastive hinge loss:

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} [\Delta - sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)) + sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{v}))]_+$$

$$[a]_+ = \max(0, a)$$

- Δ is the “margin”

(Better) Learning for Similarity

- Large-margin contrastive hinge loss:

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} [\Delta - sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)) + sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{v}))]_+$$

- How should we choose negative examples?

(Better) Learning for Similarity

- Large-margin contrastive hinge loss:

$$\min_{\boldsymbol{\theta}} \sum_{\langle \boldsymbol{x}_1, \boldsymbol{x}_2 \rangle \in \mathcal{T}} \left[\Delta - sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{x}_2)) + sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{v})) \right]_+$$

- How should we choose negative examples?
  - random: just pick *v* randomly from the data
  - max: $\boldsymbol{v} = \operatorname*{argmax}_{\boldsymbol{s} : \langle \cdot, \boldsymbol{s} \rangle \in \mathcal{T}, \boldsymbol{s} \neq \boldsymbol{x}_1} sim(f_{\boldsymbol{\theta}}(\boldsymbol{x}_1), f_{\boldsymbol{\theta}}(\boldsymbol{s}))$
  - many other ways depending on problem

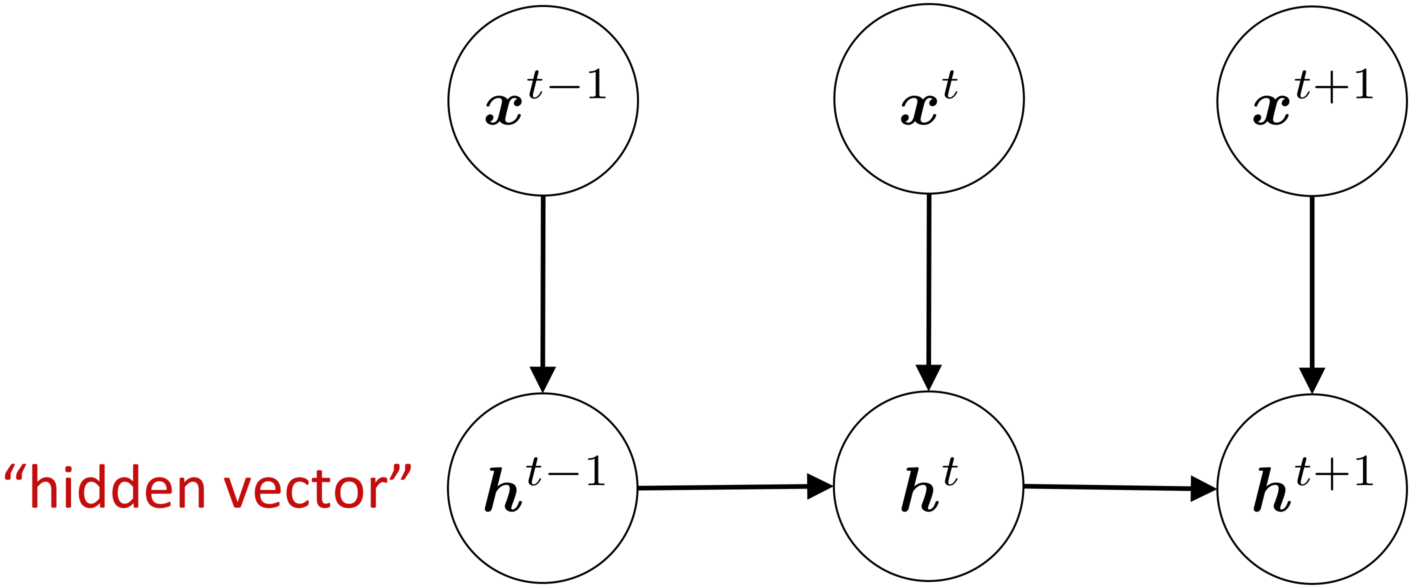
Aside:

On Multiplicative Integration with Recurrent Neural Networks

Yuhuai Wu[1,*], **Saizheng Zhang**[2,*], **Ying Zhang**[2], **Yoshua Bengio**[2,4] **and Ruslan Salakhutdinov**[3,4]
[1]University of Toronto, [2]MILA, Université de Montréal, [3]Carnegie Mellon University, [4]CIFAR
ywu@cs.toronto.edu, [2]{firstname.lastname}@umontreal.ca, rsalakhu@cs.cmu.edu

Recurrent Neural Networks

$$\boldsymbol{h}^t = \tanh\left(W^{(x)}\boldsymbol{x}^t + W^{(h)}\boldsymbol{h}^{t-1} + \boldsymbol{b}^{(h)}\right)$$



Recurrent Neural Networks

$$\boldsymbol{h}^t = \tanh\left(W^{(x)}\boldsymbol{x}^t + W^{(h)}\boldsymbol{h}^{t-1} + \boldsymbol{b}^{(h)}\right)$$

Multiplicative Integration Recurrent Neural Networks

$$\boldsymbol{h}^t = \tanh\left(W^{(x)}\boldsymbol{x}^t \odot W^{(h)}\boldsymbol{h}^{t-1} + \boldsymbol{b}^{(h)}\right)$$

On Multiplicative Integration with Recurrent Neural Networks

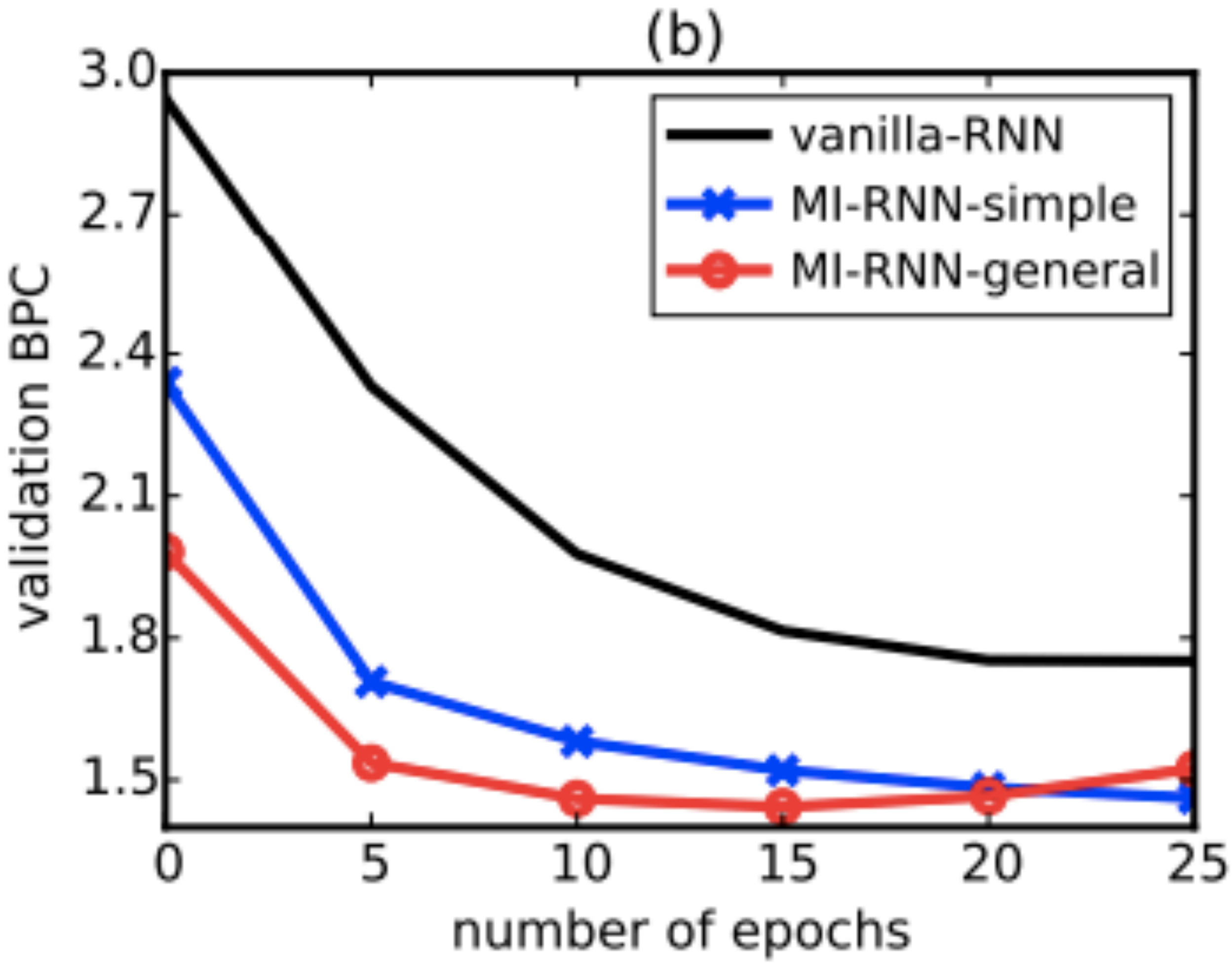
2.2 Gradient Properties

The Multiplicative Integration has different gradient properties compared to the additive building block. For clarity of presentation, we first look at vanilla-RNN and RNN with Multiplicative Integration embedded, referred to as **MI-RNN**. That is, $\boldsymbol{h}_t = \phi(\mathbf{W}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{b})$ versus $\boldsymbol{h}_t = \phi(\mathbf{W}\boldsymbol{x}_t \odot \mathbf{U}\boldsymbol{h}_{t-1} + \mathbf{b})$. In a vanilla-RNN, the gradient $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}}$ can be computed as follows:

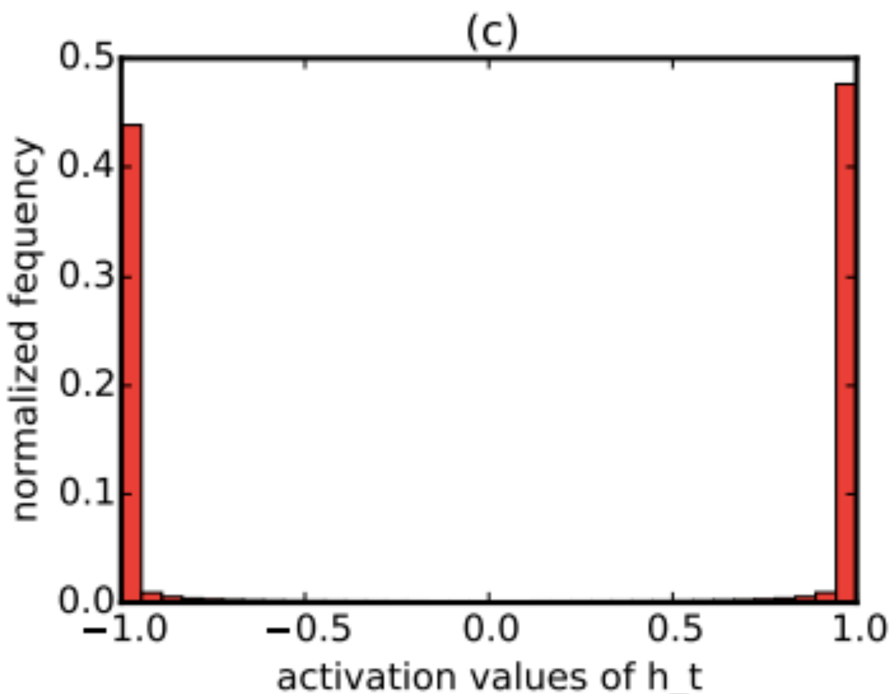
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}} = \prod_{k=t-n+1}^{t} \mathbf{U}^T \mathrm{diag}(\phi'_k), \tag{5}$$

where $\phi'_k = \phi'(\mathbf{W}\boldsymbol{x}_k + \mathbf{U}\boldsymbol{h}_{k-1} + \mathbf{b})$. The equation above shows that the gradient flow through time heavily depends on the hidden-to-hidden matrix $\mathbf{U}$, but $\mathbf{W}$ and $\boldsymbol{x}_k$ appear to play a limited role: they only come in the derivative of ϕ' mixed with $\mathbf{U}\boldsymbol{h}_{k-1}$. On the other hand, the gradient $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}}$ of a MI-RNN is[4]:

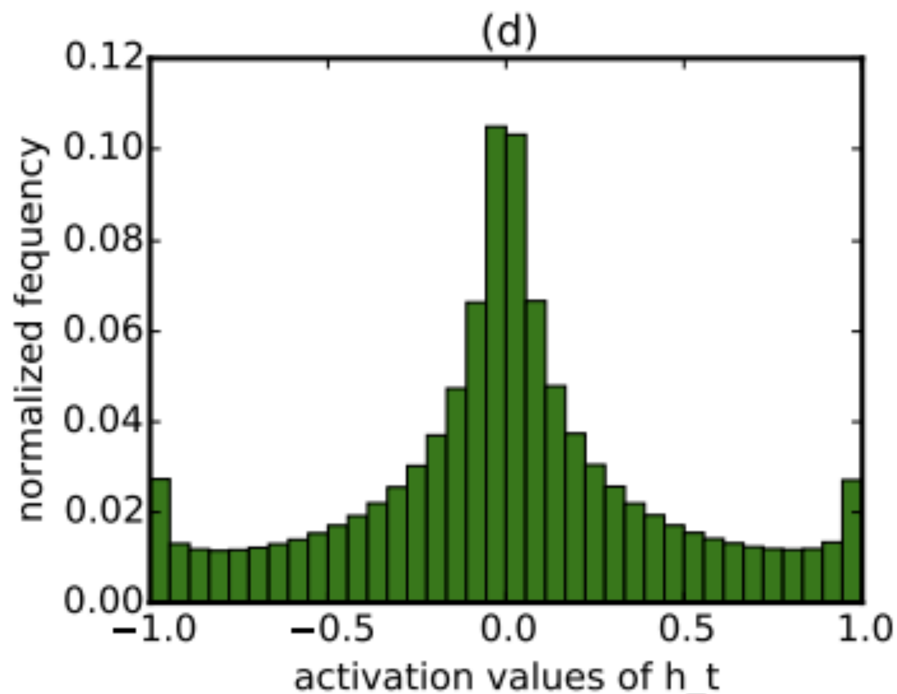
$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-n}} = \prod_{k=t-n+1}^{t} \mathbf{U}^T \mathrm{diag}(\mathbf{W}\boldsymbol{x}_k)\mathrm{diag}(\phi'_k), \tag{6}$$


(b)
vanilla-RNN
MI-RNN-simple
MI-RNN-general
validation BPC
3.0
2.7
2.4
2.1
1.8
1.5
0
5
10
15
20
25
number of epochs

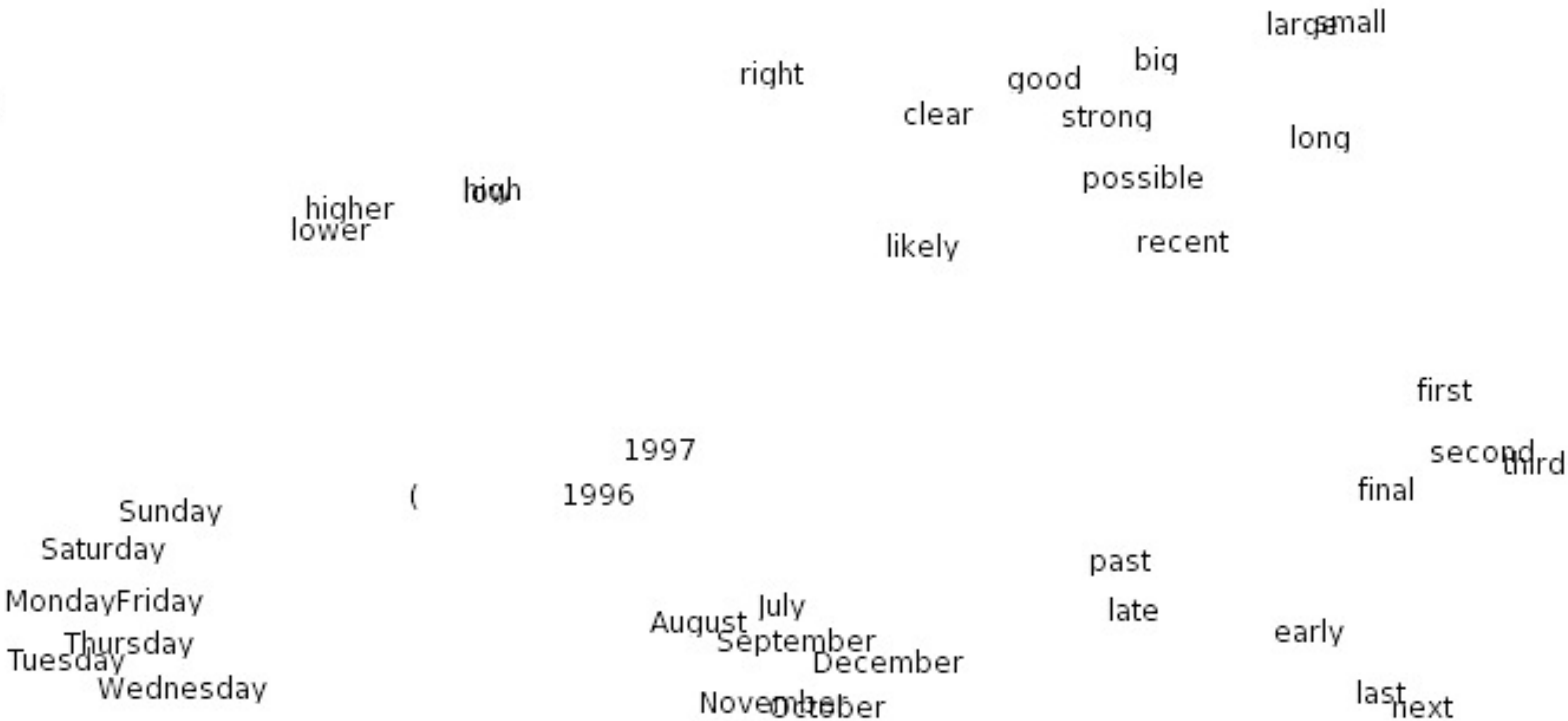
RNN


(c)
0.5
0.4
0.3
0.2
0.1
0.0
normalized fequency
−1.0
−0.5
0.0
0.5
1.0
activation values of h_t

MI-RNN


(d)
0.12
0.10
0.08
0.06
0.04
0.02
0.00
normalized fequency
−1.0
−0.5
0.0
0.5
1.0
activation values of h_t

Word Embeddings



Turian et al. (2010)

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

- idea: use a neural network for *n*-gram language modeling:

$$P_{\boldsymbol{\theta}}(w_t \mid w_{t-n+1}, \ldots, w_{t-2}, w_{t-1})$$

Journal of Machine Learning Research 3 (2003) 1137–1155 Submitted 4/02; Published 2/03

A Neural Probabilistic Language Model

Yoshua Bengio BENGIOY@IRO.UMONTREAL.CA
Réjean Ducharme DUCHARME@IRO.UMONTREAL.CA
Pascal Vincent VINCENTP@IRO.UMONTREAL.CA
Christian Jauvin JAUVINC@IRO.UMONTREAL.CA
Département d'Informatique et Recherche Opérationnelle
Centre de Recherche Mathématiques
Université de Montréal, Montréal, Québec, Canada

- this is not the earliest paper on using neural networks for *n*-gram language models, but it's the most well-known and first to scale up
- see paper for citations of earlier work

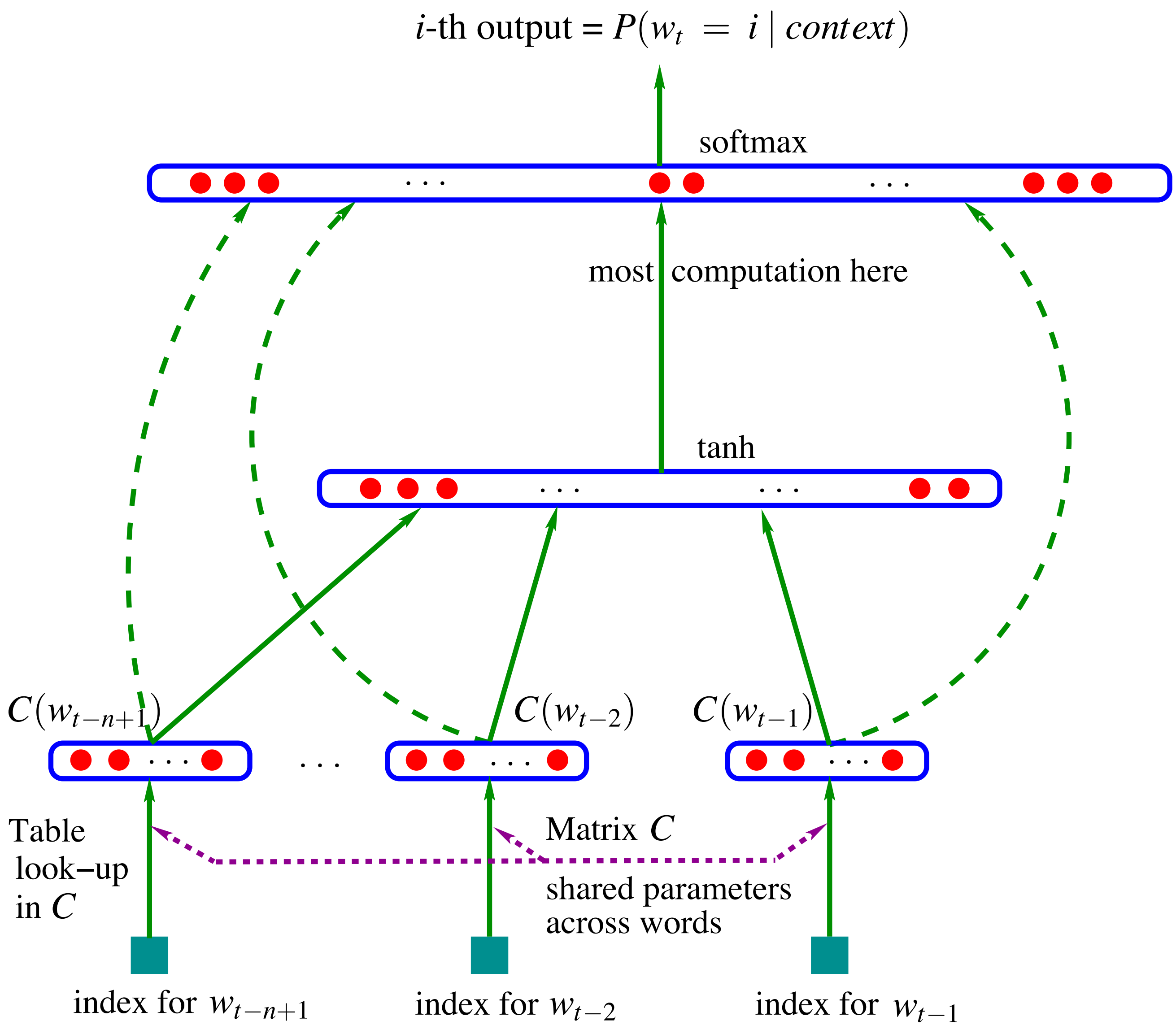
Neural Probabilistic Language Models
(Bengio et al., 2003)

1.1 Fighting the Curse of Dimensionality with Distributed Representations

In a nutshell, the idea of the proposed approach can be summarized as follows:

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

Model (Bengio et al., 2003)



Bengio et al. (2003)

- Experiments:
  - they minimized log loss of next word conditioned on a fixed number of previous words
  - no RNNs here. just a feed-forward network.
  - ~800k training tokens, vocab size of 17k
  - they trained for 5 epochs, which took 3 weeks on 40 CPUs!

Experiments (Bengio et al., 2003)

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**

classes). n : order of the model. c : number of word classes in class-based n-grams. h : number of hidden units. m : number of word features for MLPs, number of classes for class-based n-grams. *direct*: whether there are direct connections from word features to outputs. *mix*: whether the output probabilities of the neural network are mixed with the output of the trigram (with a weight of 0.5 on each). The last three columns give perplexity on the training, validation and test sets.

Experiments (Bengio et al., 2003)

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**

- Observations:
  - hidden layer (h > 0) helps
  - interpolating with n-gram model (“mix”) helps
  - using higher word embedding dimensionality helps
  - 5-gram model better than trigram

Experiments

	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321
class-based back-off	3	150						348	334
class-based back-off	3	200						354	340
class-based back-off	3	500						326	**312**
class-based back-off	3	1000						335	319

Bengio et al. (2003)

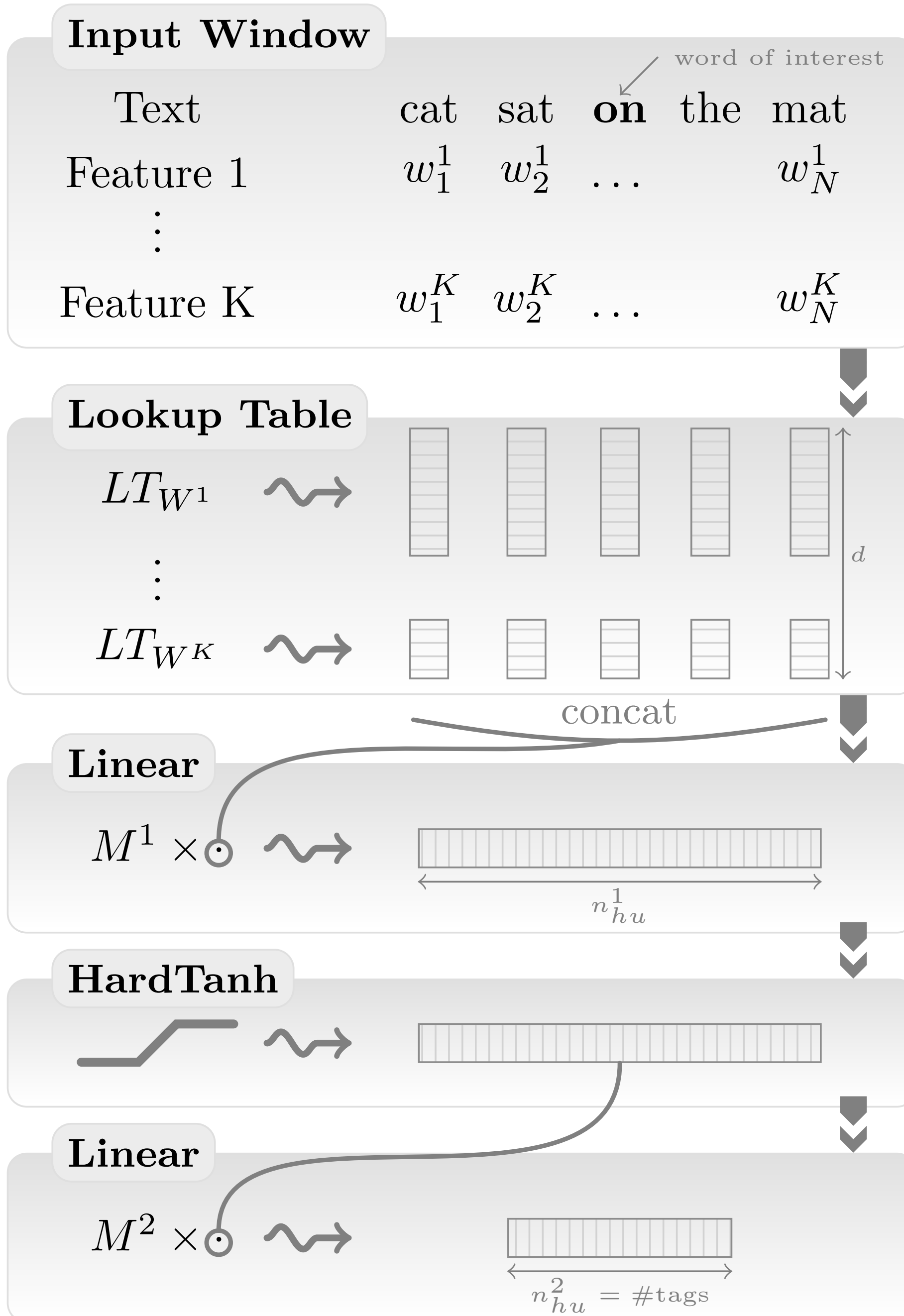
- they discuss how the word embedding space might be interesting to examine but they don't do this
- they suggest that a good way to visualize/ interpret word embeddings would be to use 2 dimensions ☺
- they discussed handling polysemous words, unknown words, inference speed-ups, etc.

Collobert et al. (2011)

Journal of Machine Learning Research 12 (2011) 2493-2537 Submitted 1/10; Revised 11/10; Published 8/11

Natural Language Processing (Almost) from Scratch

Ronan Collobert[*] RONAN@COLLOBERT.COM
Jason Weston[†] JWESTON@GOOGLE.COM
Léon Bottou[‡] LEON@BOTTOU.ORG
Michael Karlen MICHAEL.KARLEN@GMAIL.COM
Koray Kavukcuoglu[§] KORAY@CS.NYU.EDU
Pavel Kuksa[¶] PKUKSA@CS.RUTGERS.EDU
NEC Laboratories America
4 Independence Way
Princeton, NJ 08540



Input Window
word of interest
Text
cat sat on the mat
Feature 1
w_1^1 w_2^1 ... w_N^1
Feature K
w_1^K w_2^K ... w_N^K
Lookup Table
LT_{W^1}
LT_{W^K}
d
concat
Linear
$M^1 \times \odot$
n_{hu}^1
HardTanh
Linear
$M^2 \times \odot$
$n_{hu}^2 = \#\text{tags}$

Collobert et al. Pairwise Ranking Loss

$$\min_{\boldsymbol{\theta}} \sum_{\langle x_1,...,x_{11}\rangle \in \mathcal{T}} \sum_{w \in \mathcal{V}} [1 - f_{\boldsymbol{\theta}}(\langle x_1, ..., x_{11}\rangle) + f_{\boldsymbol{\theta}}(\langle x_1, ..., x_5, w, x_7, ..., x_{11}\rangle)]_+$$

- $\mathcal{T}$ is training set of 11-word windows
- $\mathcal{V}$ is vocabulary
- What is going on here? (loss C on handout)

Collobert et al. Pairwise Ranking Loss

$$\min_{\boldsymbol{\theta}} \sum_{\langle x_1,\ldots,x_{11}\rangle \in \mathcal{T}} \sum_{w \in \mathcal{V}} [1 - f_{\boldsymbol{\theta}}(\langle x_1, \ldots, x_{11}\rangle) + f_{\boldsymbol{\theta}}(\langle x_1, \ldots, x_5, w, x_7, \ldots, x_{11}\rangle)]_+$$

- $\mathcal{T}$ is training set of 11-word windows
- $\mathcal{V}$ is vocabulary
- What is going on here?
  - Make actual text window have higher score than all windows with center word replaced by *w*

Collobert et al. Pairwise Ranking Loss

$$\min_{\boldsymbol{\theta}} \sum_{\langle x_1,\ldots,x_{11}\rangle \in \mathcal{T}} \sum_{w \in \mathcal{V}} [1 - f_{\boldsymbol{\theta}}(\langle x_1, ..., x_{11}\rangle) + f_{\boldsymbol{\theta}}(\langle x_1, ..., x_5, w, x_7, ..., x_{11}\rangle)]_+$$

- $\mathcal{T}$ is training set of 11-word windows
- $\mathcal{V}$ is vocabulary
- This still sums over entire vocabulary, so it should be as slow as log loss…
- Why can it be faster?
  - when using SGD, summation → sample