Natural Language Processing with Deep Learning
CS224N/Ling284

Lecture 8: Recurrent Neural Networks

Christopher Manning and Richard Socher

Organization

Extra project office hour today after lecture

Overview

- Traditional language models
- RNNs
- RNN language models
- Important training problems and tricks
  - Intuition for vanishing gradient problem with toy example
  - Vanishing and exploding gradient problems
- RNNs for other sequence tasks
- Bidirectional and deep RNNs

Language Models

A language model computes a probability for a sequence of words: $P(w_1, \ldots, w_T)$

- Useful for machine translation
  - Word ordering:
    p(the cat is small) > p(small the is cat)
  - Word choice:
    p(walking home after school) > p(walking house after school)

Traditional Language Models

- Probability is usually conditioned on window of n previous words
- An incorrect but necessary Markov assumption!

$$P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1})$$

- To estimate probabilities, compute for unigrams and bigrams (conditioning on one/two previous word(s):

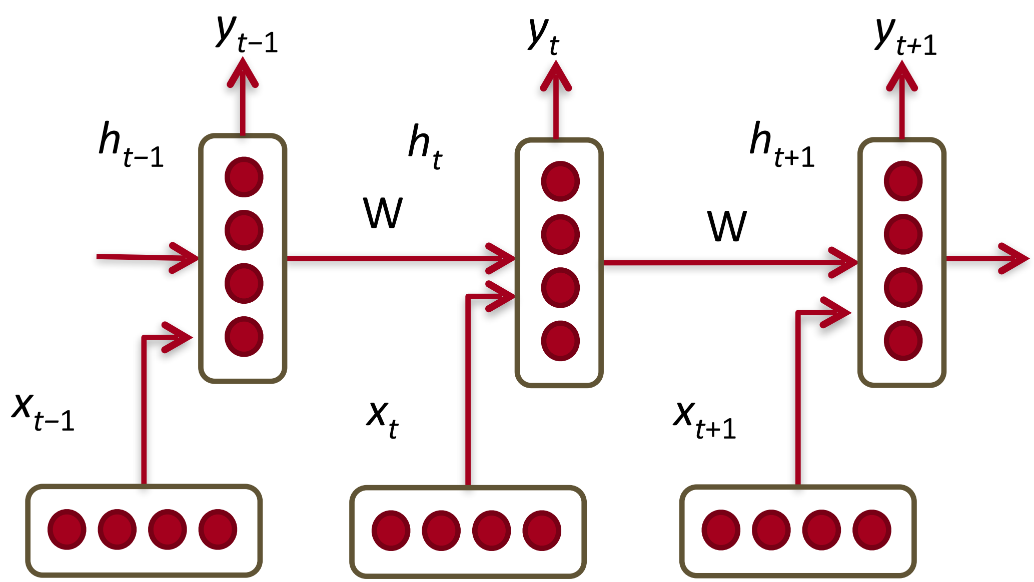
$$p(w_2|w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)} \qquad p(w_3|w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$$

Traditional Language Models

- Performance improves with keeping around higher n-grams counts and doing smoothing and so-called backoff (e.g. if 4-gram not found, try 3-gram, etc)
- There are A LOT of n-grams!
  → Gigantic RAM requirements!
- Recent state of the art: *Scalable Modified Kneser-Ney Language Model Estimation* by Heafield et al.:
  "Using one machine with 140 GB RAM for 2.8 days, we built an unpruned model on 126 billion tokens"

Recurrent Neural Networks!

- RNNs tie the weights at each time step
- Condition the neural network on all previous words
- RAM requirement only scales with number of words



Recurrent Neural Network language model

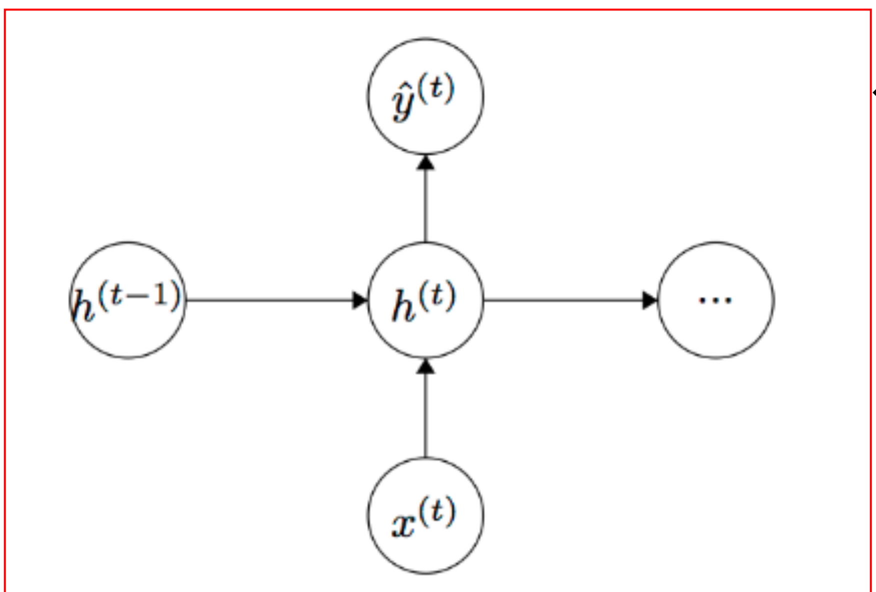
Given list of word **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$

At a single time step:

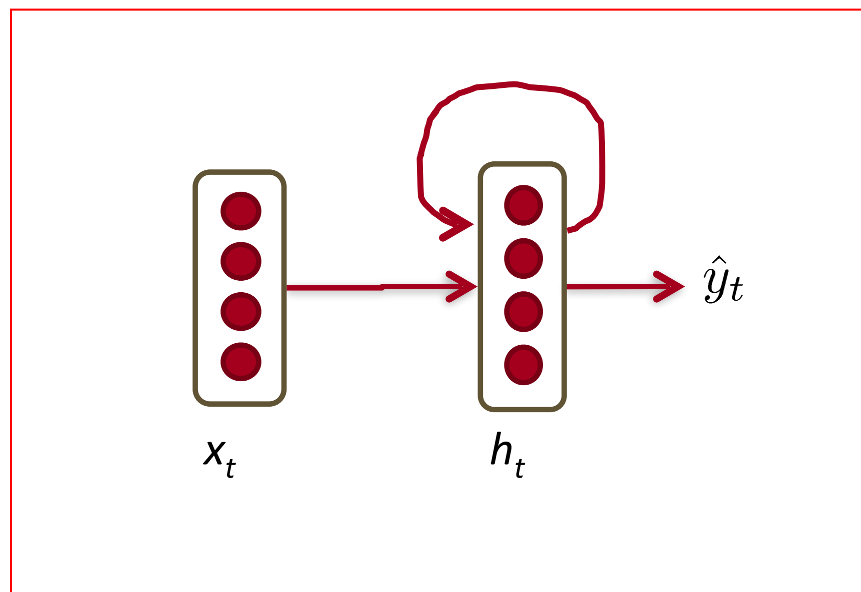
$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$

$$\hat{y}_t = \mathrm{softmax}\left(W^{(S)} h_t\right)$$

$$\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) = \hat{y}_{t,j}$$



$\leftrightarrow$



Recurrent Neural Network language model

Main idea: we use the same set of W weights at all time steps!

Everything else is the same:

$$
\begin{aligned}
h_t &= \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right) \\
\hat{y}_t &= \mathrm{softmax}\left(W^{(S)} h_t\right) \\
\hat{P}(x_{t+1} = v_j \mid x_t, \ldots, x_1) &= \hat{y}_{t,j}
\end{aligned}
$$

$h_0 \in \mathbb{R}^{D_h}$ is some initialization vector for the hidden layer at time step 0

$x_{[t]}$ is the column vector of L at index [t] at time step t

$W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$ $\quad W^{(hx)} \in \mathbb{R}^{D_h \times d}$ $\quad W^{(S)} \in \mathbb{R}^{|V| \times D_h}$

Recurrent Neural Network language model

$\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

Recurrent Neural Network language model

Evaluation could just be negative of average log probability over dataset of size (number of words) T:

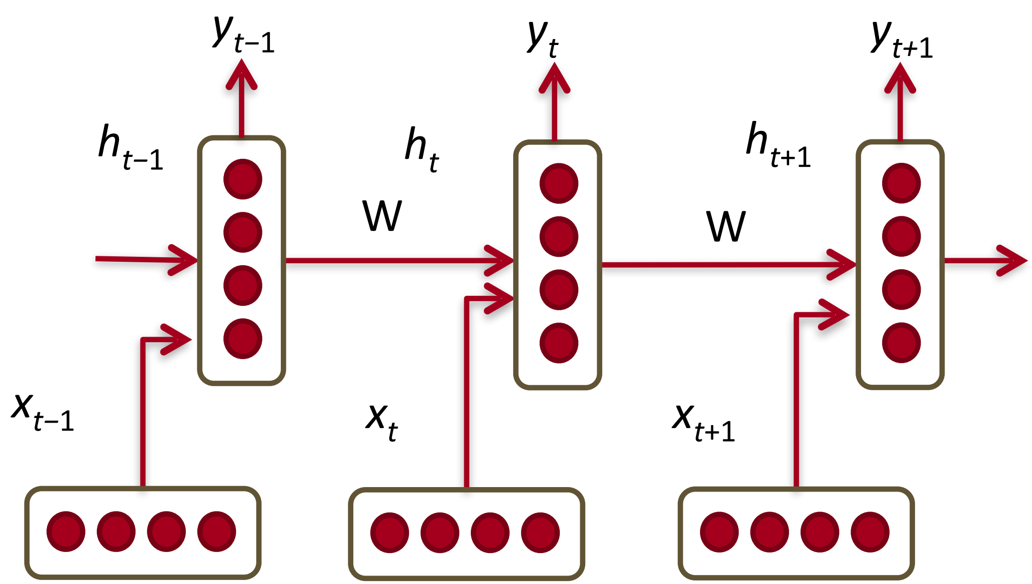
$$J = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j}\log \hat{y}_{t,j}$$

But more common: Perplexity: 2^J

Lower is better!

Training RNNs is hard

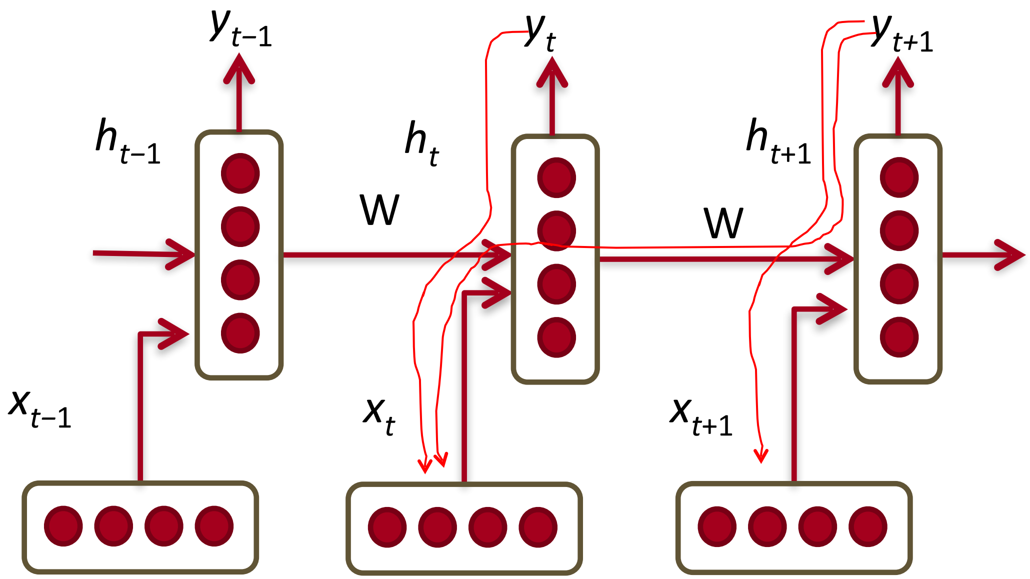
- Multiply the same matrix at each time step during forward prop



- Ideally inputs from many time steps ago can modify output *y*
- Take $\frac{\partial E_2}{\partial W}$ for an example RNN with 2 time steps! Insightful!

The vanishing/exploding gradient problem

- Multiply the same matrix at each time step during backprop



The vanishing gradient problem - Details

- Similar but simpler RNN formulation:

$$\begin{aligned} h_t &= W f(h_{t-1}) + W^{(hx)} x_{[t]} \\ \hat{y}_t &= W^{(S)} f(h_t) \end{aligned}$$

- Total error is the sum of each error at time steps t

$$\frac{\partial E}{\partial W} = \sum_{t=1}^{T} \frac{\partial E_t}{\partial W}$$

- Hardcore chain rule application:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$$

The vanishing gradient problem - Details

- Similar to backprop but less efficient formulation
- Useful for analysis we'll look at:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$$

- Remember: $h_t = W f(h_{t-1}) + W^{(hx)} x_{[t]}$
- More chain rule, remember:

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}}$$

- Each partial is a Jacobian:

$$\frac{d\mathbf{f}}{d\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x_1} \cdots \frac{\partial \mathbf{f}}{\partial x_n} \right] = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The vanishing gradient problem - Details

- Analyzing the norms of the Jacobians, yields:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \|W^T\| \|\mathrm{diag}[f'(h_{j-1})]\| \leq \beta_W \beta_h$$

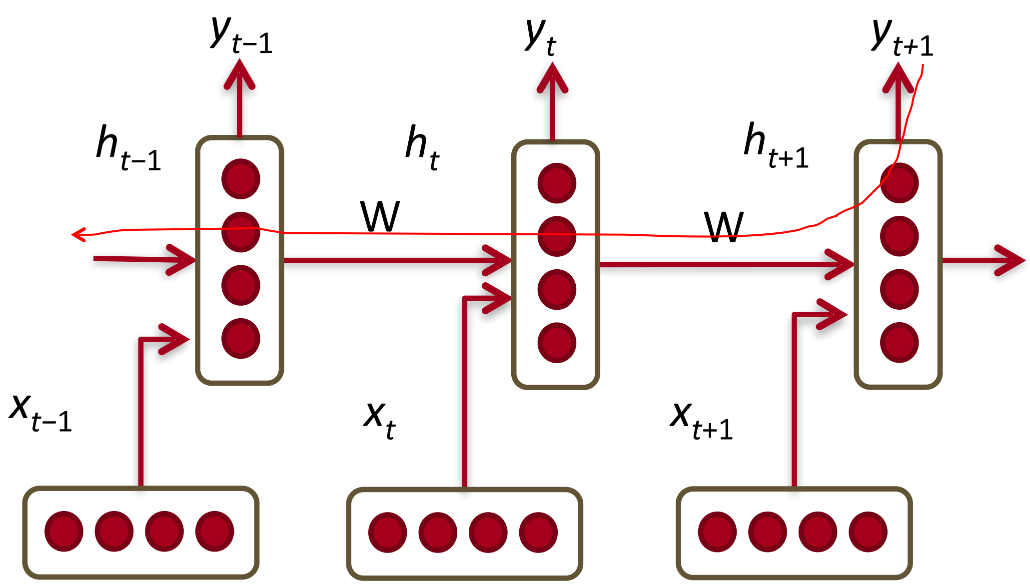
- Where we defined β's as upper bounds of the norms
- The gradient is a product of Jacobian matrices, each associated with a step in the forward computation.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_W \beta_h)^{t-k}$$

- This can become very small or very large quickly [Bengio et al 1994], and the locality assumption of gradient descent breaks down. → **Vanishing or exploding gradient**

Why is the vanishing gradient a problem?

- The error at a time step ideally can tell a previous time step from many steps away to change during backprop



The vanishing gradient problem for language models

- In the case of language modeling or question answering words from time steps far away are not taken into consideration when training to predict the next word

- Example:

  Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____

Structured Training for Neural Network Transition-Based Parsing

David Weiss, Chris Alberti, Michael Collins, Slav Petrov

Presented by: Shayne Longpre

2/2/17

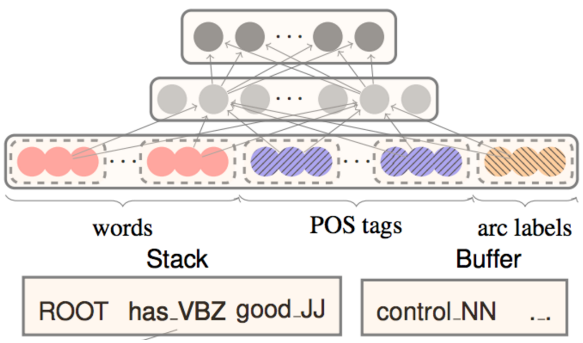
What is SyntaxNet?

- 2016/5: Google announces the “World’s Most Accurate Parser Goes Open Source”
- SyntaxNet (2016): New, fast, performant Tensorflow framework for syntactic parsing.
- Now supports 40 languages -- Parse McParseface’s 40 ‘cousins’

What is SyntaxNet?

- 2016/5: Google announces the “World’s Most Accurate Parser Goes Open Source”
- SyntaxNet (2016): New, fast, performant Tensorflow framework for syntactic parsing.
- Now supports 40 languages -- Parse McParseface’s 40 “cousins”

Chen & Manning (2014) → Weiss et al. (2015) → Andor et al. (2016)



\+ Unlabelled Data
\+ Tune Model
\+ Structured Perceptron & Beam Search

\+ Global Normalization

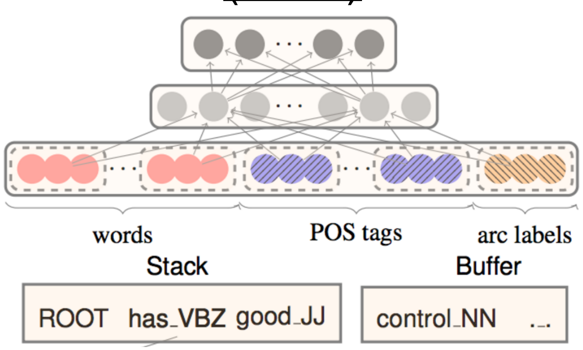
=

SyntaxNet

What is SyntaxNet?

- 2016/5: Google announces the "World's Most Accurate Parser Goes Open Source"
- SyntaxNet (2016): New, fast, performant Tensorflow framework for syntactic parsing.
- Now supports 40 languages -- Parse McParseface's 40 'cousins'

Chen & Manning (2014)



Weiss et al. (2015)

+ Unlabelled Data
+ Tune Model
+ Structured Perceptron & Beam Search

Andor et al. (2016)

+ Global Normalization

=

SyntaxNet

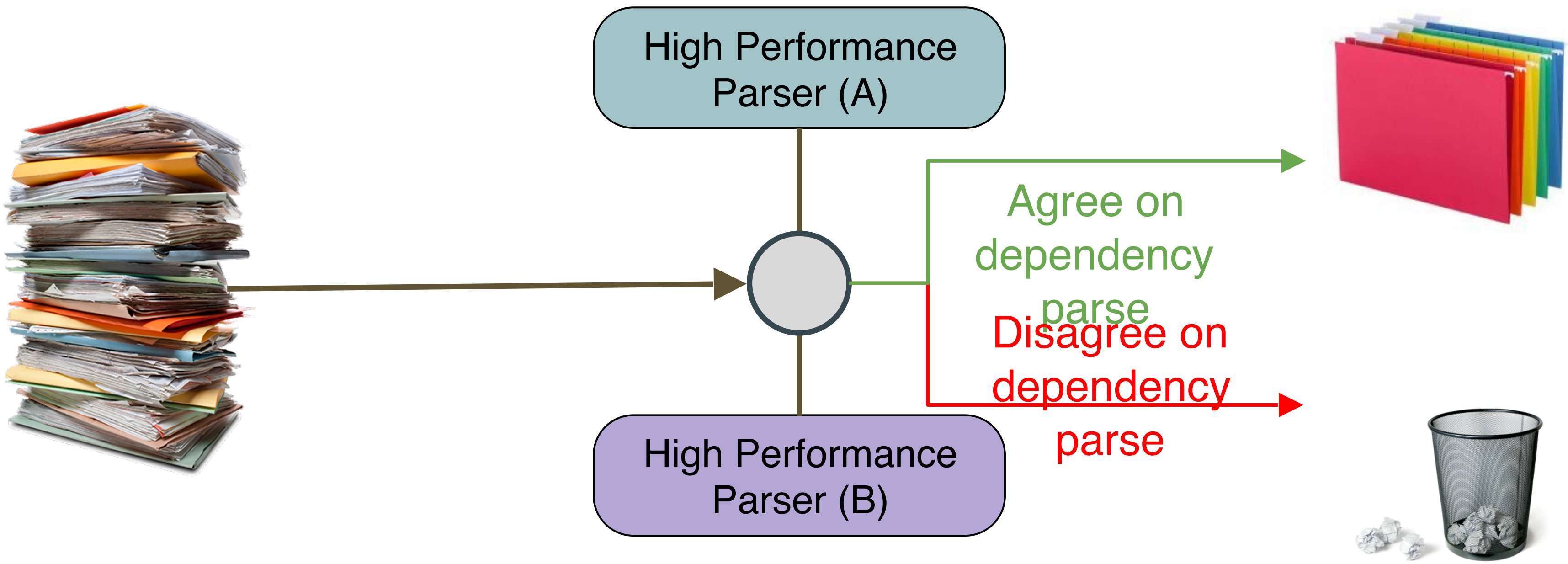
3 New Contributions (... since Q2 in Assignment 2)

1. Leverage Unlabelled Data -- "Tri-Training"

2. Tuned Neural Network Model

3. Final Layer: Structured Perceptron w/ Beam Search

1. Tri-Training: *Leverage Unlabelled Data*



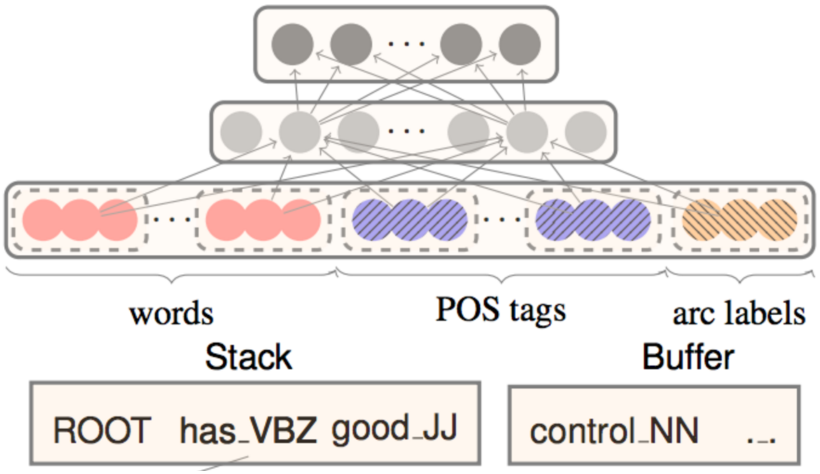
Unlabelled Data

"Tri-Training" (Li et al, 2014)

Labelled Data

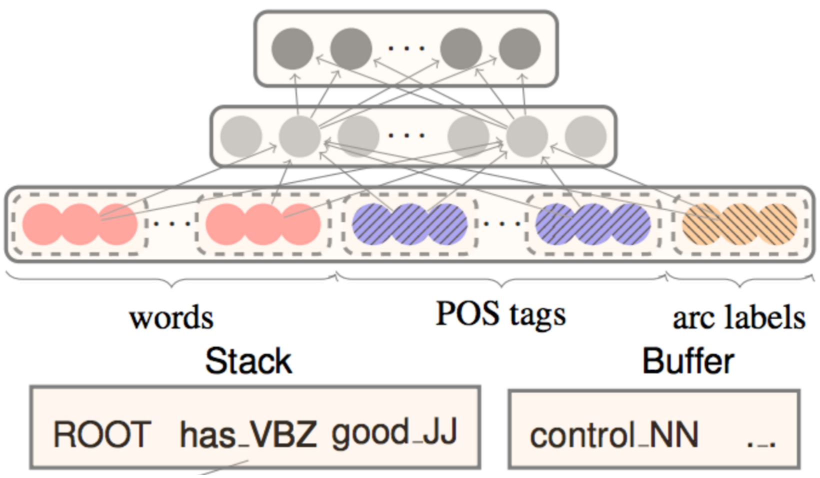
2. Model Changes

Chen & Manning (2014):

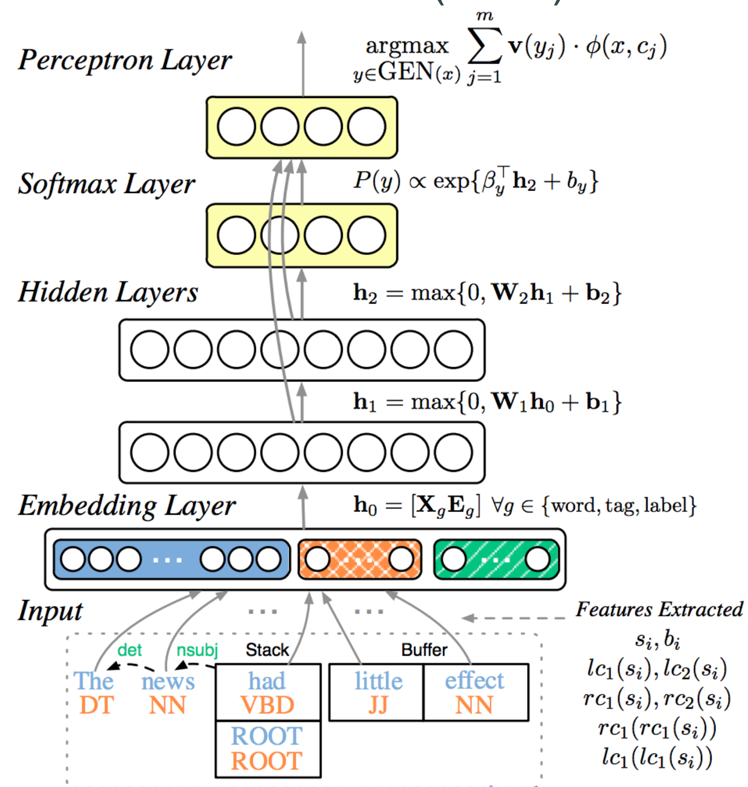


2. Model Changes

Chen & Manning (2014):

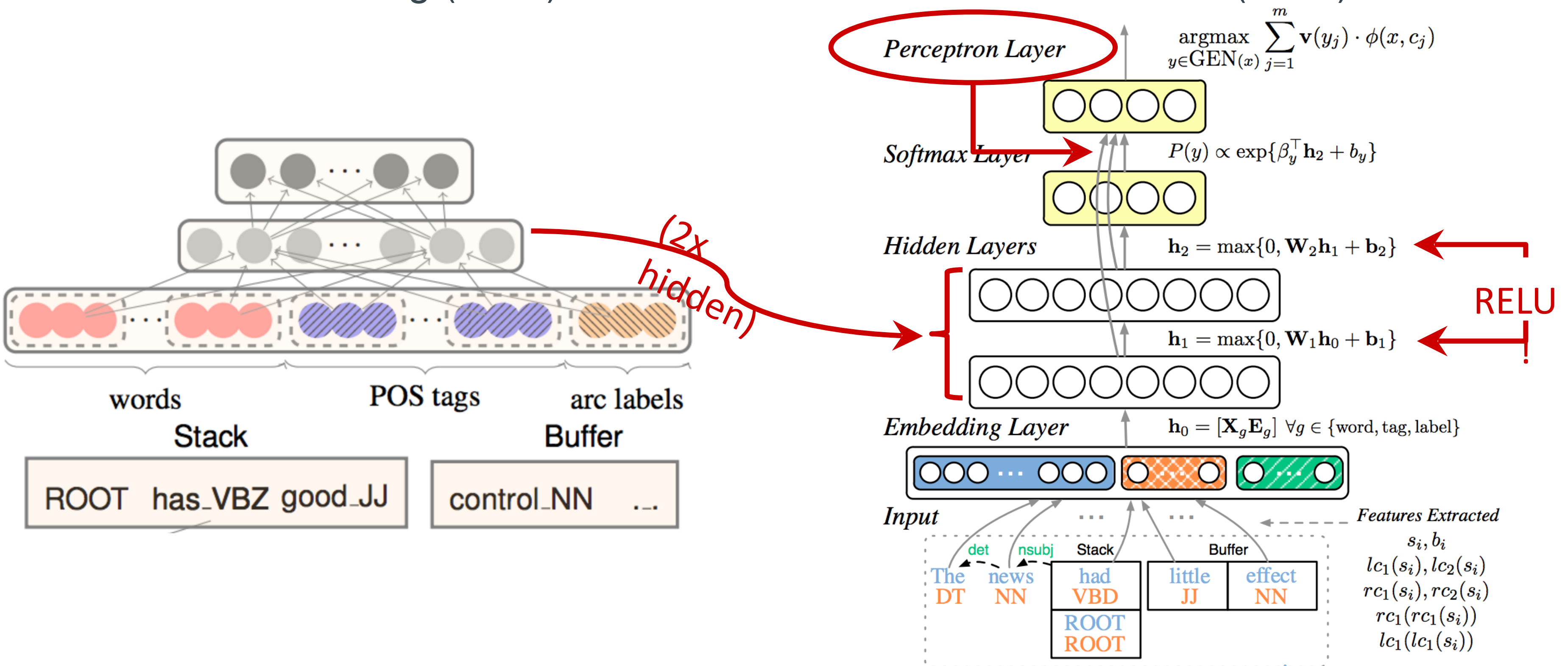


Weiss et al. (2015):



2. Model Changes

Chen & Manning (2014): Weiss et al. (2015):



3. Structured Perceptron Training + Beam Search

Problem: Greedy algorithms are unable to look beyond one step ahead, or recover from incorrect decisions.

3. Structured Perceptron Training + Beam Search

Problem: Greedy algorithms are unable to look beyond one step ahead, or recover from incorrect decisions.

Solution: Look forward -- search the tree of possible transition sequences.

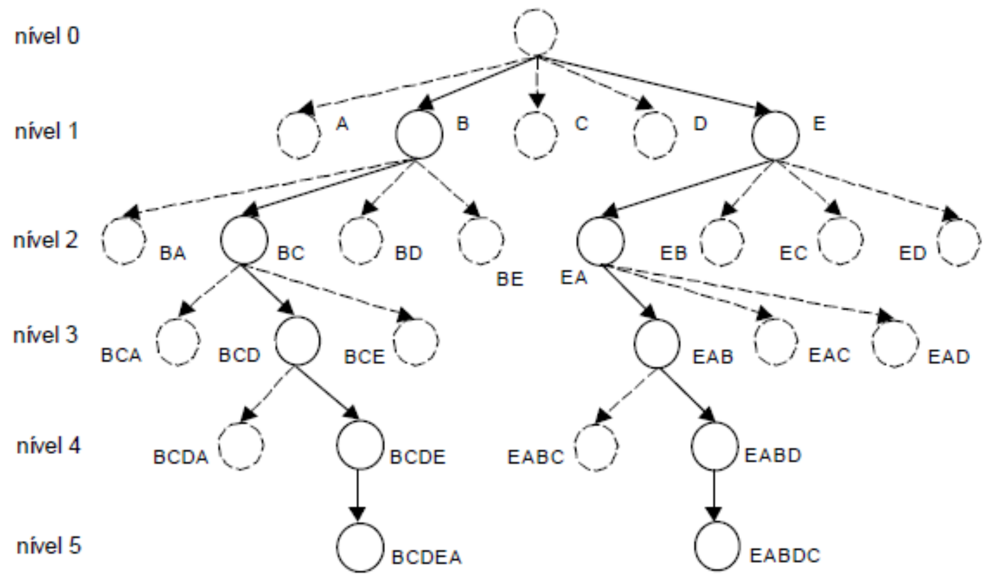


Figura 1 – Árvore de busca utilizando o *beam search*

3. Structured Perceptron Training + Beam Search

Problem: Greedy algorithms are unable to look beyond one step ahead, or recover from incorrect decisions.

Solution: Look forward -- search the tree of possible transition sequences.

- Keep track of *K* top partial transition sequences up to depth *m*.
- Score transition using perceptron:

$$\operatorname*{argmax}_{y \in \mathrm{GEN}(x)} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \phi(x, y_1 \ldots y_{j-1}).$$

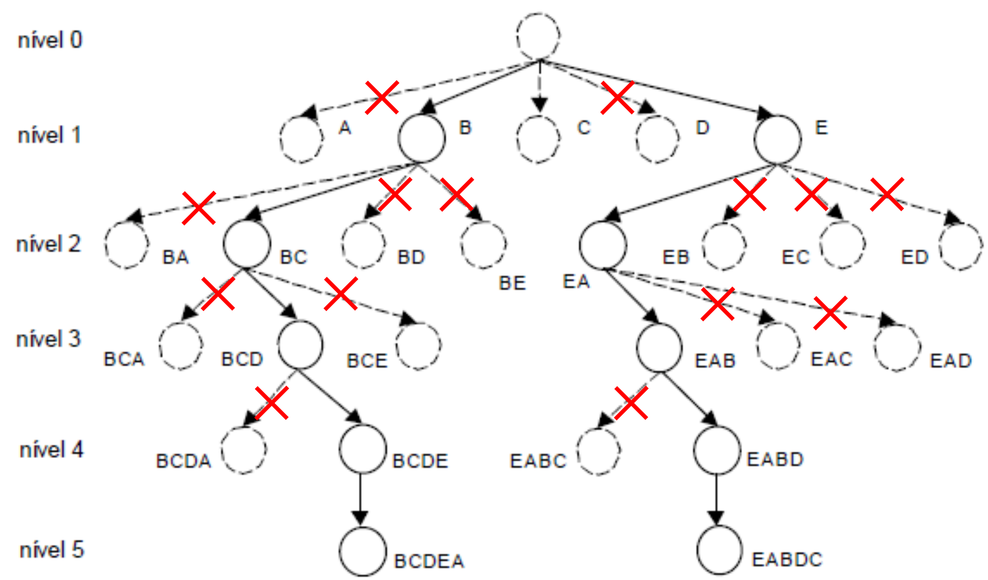


Figura 1 – Árvore de busca utilizando o *beam search*

3. Structured Perceptron Training + Beam Search

Problem: Greedy algorithms are unable to look beyond one step ahead, or recover from incorrect decisions.

Solution: Look forward -- search the tree of possible transition sequences.

- Keep track of *K* top partial transition sequences up to depth *m*.
- Score transition using perceptron:

$$\underbrace{\underset{y\in \mathrm{GEN}(x)}{\operatorname{argmax}}}_{\text{Possible transition sequences}} \sum_{j=1}^{m} \mathbf{v}(y_j) \cdot \overbrace{\phi(x, y_1 \dots y_{j-1})}^{\text{Feature vector}}.$$

Perceptron parameter vector

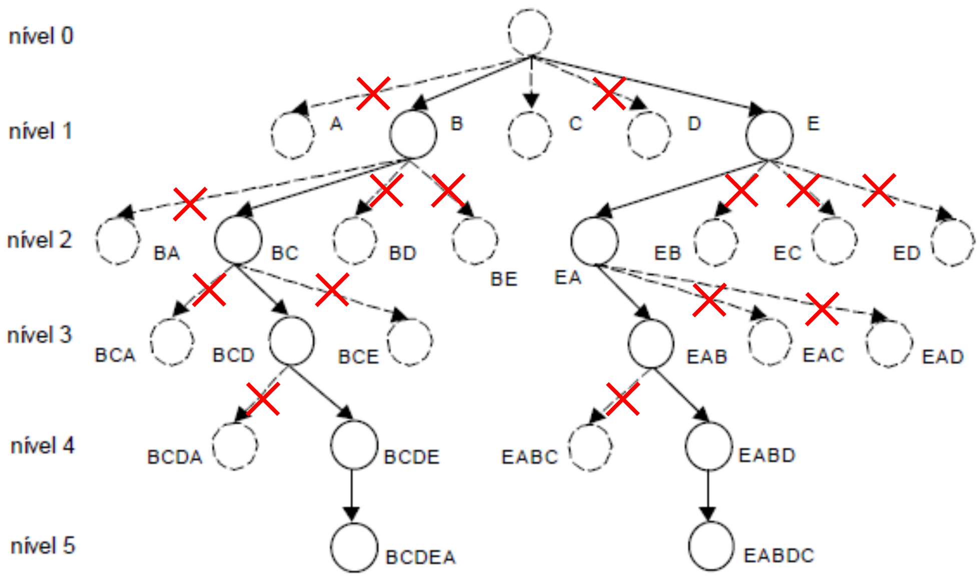


Figura 1 – Árvore de busca utilizando o *beam search*

Conclusions

Method	UAS	LAS (PTB WSJ SD 3.3
Chen & Manning 2014	92.0	89.7
Weiss et al. 2015	93.99	92.05
Andor et al. 2016	94.61	92.79

- Identify specific flaws in existing models (greedy algorithms) and solve them. In this case, with:
  - More data
  - Better tuning
  - Structured perceptron and beam search

- Final step to **SyntaxNet**: Andor et al. (2016) solve the “Label Bias Problem” using Global Normalization

IPython Notebook with vanishing gradient example

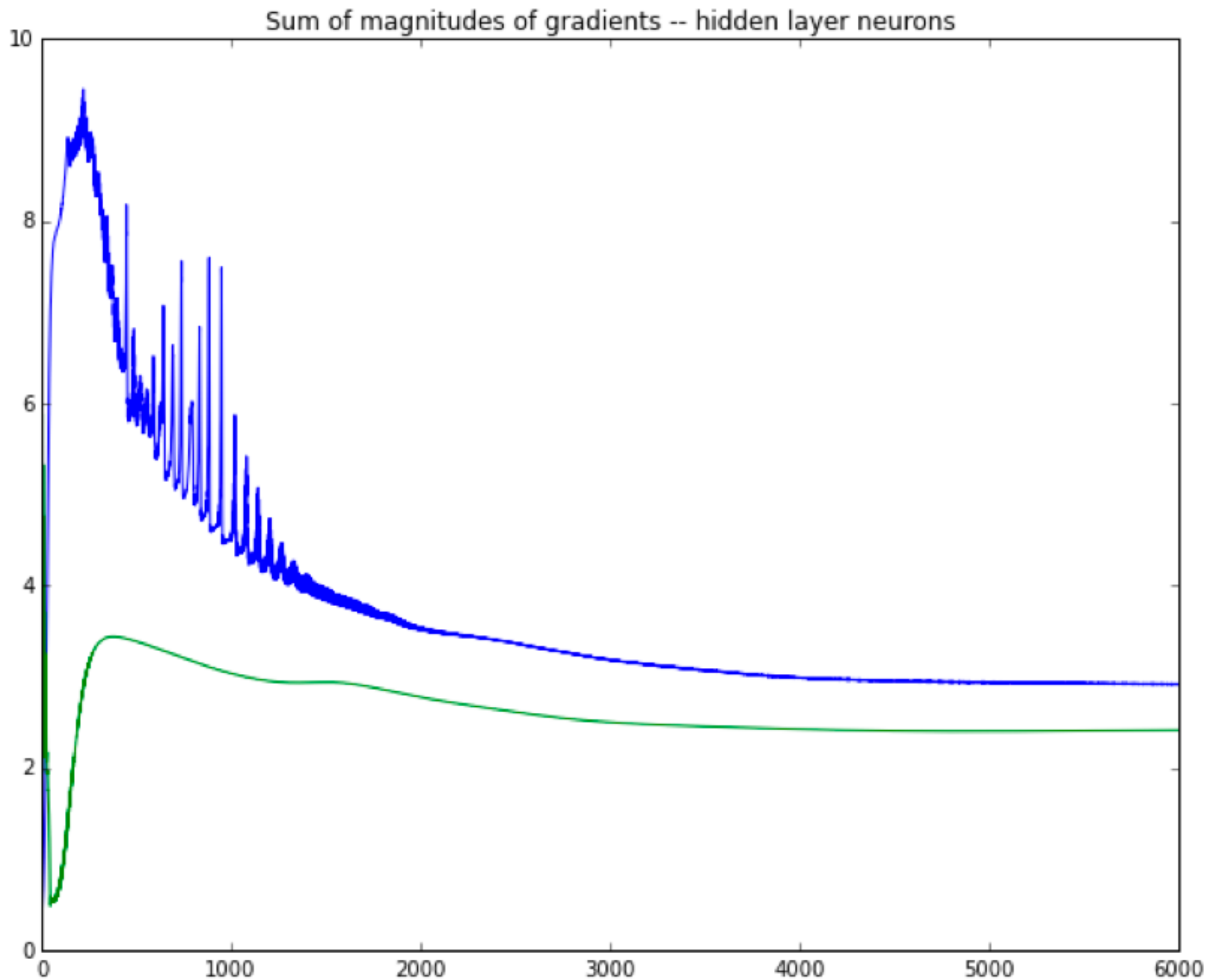
- Example of simple and clean NNet implementation
- Comparison of sigmoid and ReLu units
- A little bit of vanishing gradient

$$\boxed{\delta^{(l)} = \left((W^{(l)})^T\delta^{(l+1)}\right) \circ f'(z^{(l)}),}$$

$$\boxed{\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)}(a^{(l)})^T + \lambda W^{(l)}}$$

In [21]:
```
plt.plot(np.array(relu_array[:6000]),color='blue')
plt.plot(np.array(sigm_array[:6000]),color='green')
plt.title('Sum of magnitudes of gradients -- hidden layer neurons')
```

Out[21]: <matplotlib.text.Text at 0x10a331310>



Trick for exploding gradient: clipping trick

- The solution first introduced by Mikolov is to clip gradients to a maximum value.

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

- Makes a big difference in RNNs.

Gradient clipping intuition

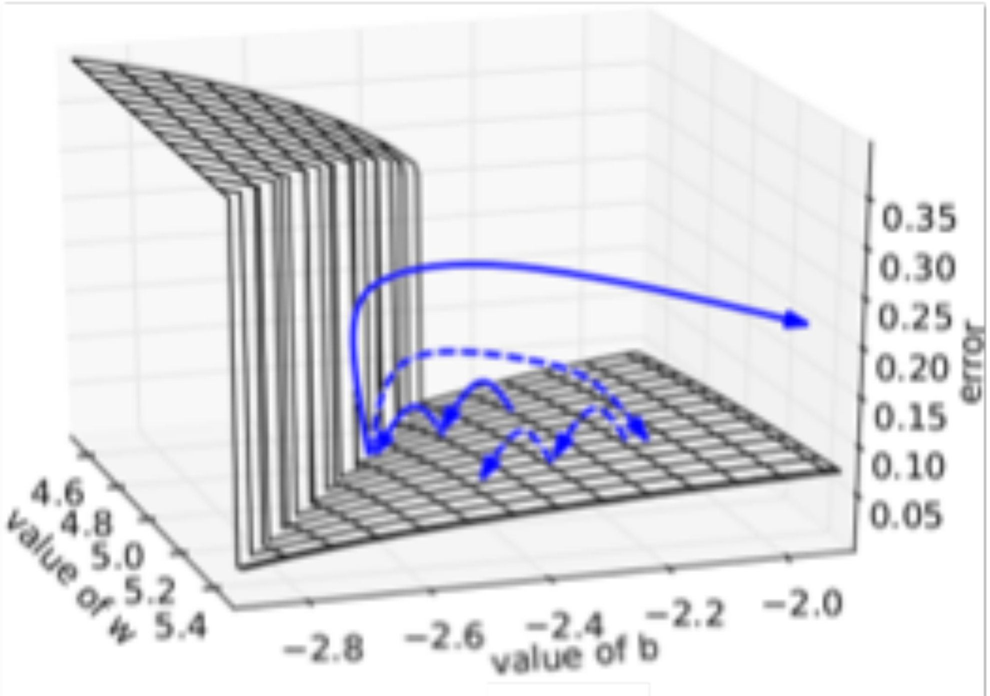
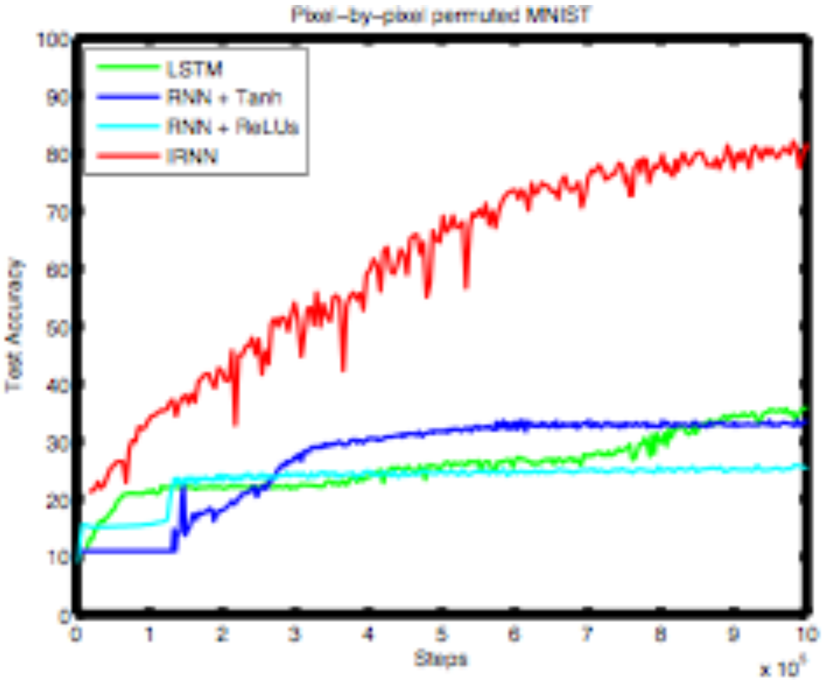


Figure from paper:
On the difficulty of training Recurrent Neural Networks, Pascanu et al. 2013

- Error surface of a single hidden unit RNN,
- High curvature walls
- Solid lines: standard gradient descent trajectories
- Dashed lines gradients rescaled to fixed size

For vanishing gradients: Initialization + ReLus!

- Initialize $W^{(*)}$'s to identity matrix I
  and
  f(z) $= \mathrm{rect}(z) = \max(z, 0)$
- → Huge difference!



- Initialization idea first introduced in *Parsing with Compositional Vector Grammars*, Socher et al. 2013

- New experiments with recurrent neural nets in *A Simple Way to Initialize Recurrent Networks of Rectified Linear* Units, Le et al. 2015

Perplexity Results

KN5 = Count-based language model with Kneser-Ney smoothing & 5-grams

Table 2. *Comparison of different neural network architectures on Penn Corpus (1M words) and Switchboard (4M words).*

	Penn Corpus		Switchboard	
Model	NN	NN+KN	NN	NN+KN
KN5 (baseline)	-	141	-	92.9
feedforward NN	141	118	85.1	77.5
RNN trained by BP	137	113	81.3	75.4
RNN trained by BPTT	123	106	77.5	72.5

Table from paper *Extensions of recurrent neural network language model* by Mikolov et al 2011

Problem: Softmax is huge and slow

Trick: Class-based word prediction

$$p(w_t|\text{history}) = p(c_t|\text{history})p(w_t|c_t)$$

$$= p(c_t|h_t)p(w_t|c_t)$$

Table 3. *Perplexities on Penn corpus with factorization of the output layer by the class model. All models have the same basic configuration (200 hidden units and BPTT=5). The Full model is a baseline and does not use classes, but the whole 10K vocabulary.*

The more classes, the better perplexity but also worse speed:

Classes	RNN	RNN+KN5	Min/epoch	Sec/test
30	134	112	12.8	8.8
50	136	114	9.8	6.7
100	136	114	9.1	5.6
200	136	113	9.5	6.0
400	134	112	10.9	8.1
1000	131	111	16.1	15.7
2000	128	109	25.3	28.7
4000	127	108	44.4	57.8
6000	127	109	70	96.5
8000	124	107	107	148
Full	123	106	154	212

One last implementation trick

- You only need to pass backwards through your sequence once and accumulate all the deltas from each E_t

Sequence modeling for other tasks

- Classify each word into:
    - NER
    - Entity level sentiment in context
    - opinionated expressions

- Example application and slides from paper *Opinion Mining with Deep Recurrent Nets* by Irsoy and Cardie 2014

Opinion Mining with Deep Recurrent Nets

Goal: Classify each word as

direct subjective expressions (DSEs) and
expressive subjective expressions (ESEs).

DSE: Explicit mentions of private states or speech events expressing private states

ESE: Expressions that indicate sentiment, emotion, etc. without explicitly conveying them.

Example Annotation

In BIO notation (tags either begin-of-entity (B_*X*) or continuation-of-entity (I_*X*)):

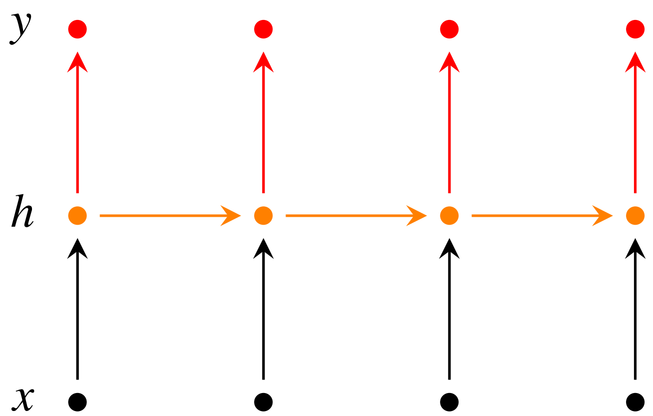
The committee, [as usual]$_{ESE}$, [has refused to make any statements]$_{DSE}$.

The	committee	,	as	usual	,	has
O	O	O	B_ESE	I_ESE	O	B_DSE

refused	to	make	any	statements	.
I_DSE	I_DSE	I_DSE	I_DSE	I_DSE	O

Approach: Recurrent Neural Network

- Notation from paper (so you get used to different ones)

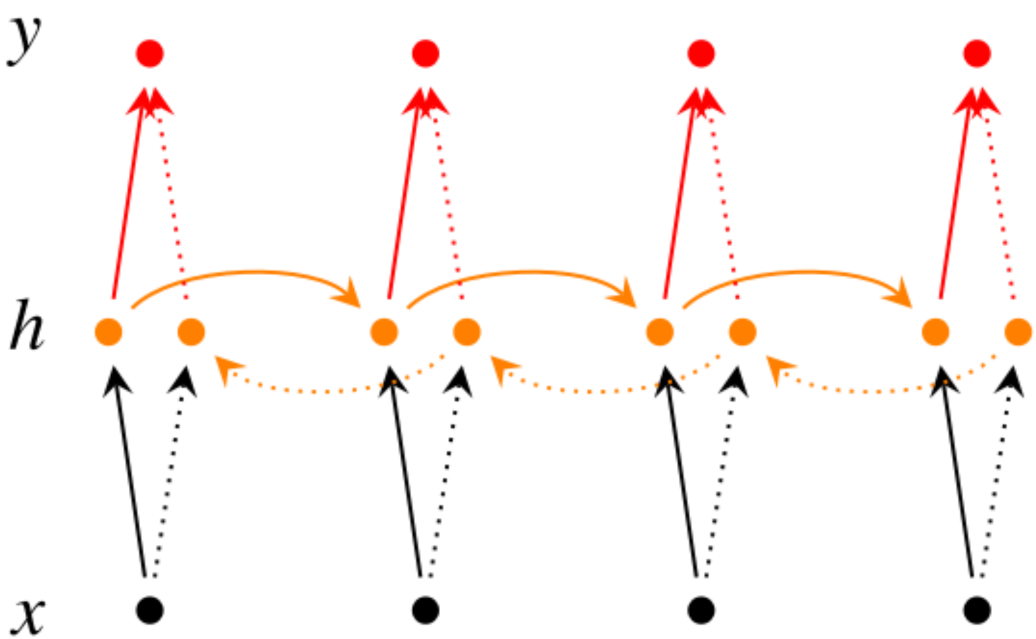


$$h_t = f(Wx_t + Vh_{t-1} + b)$$
$$y_t = g(Uh_t + c)$$

- *x* represents a token (word) as a vector.
- *y* represents the output label (B, I or O) – g = softmax !
- *h* is the memory, computed from the past memory and current word. It summarizes the sentence up to that time.

Bidirectional RNNs

Problem: For classification you want to incorporate information from words both preceding and following



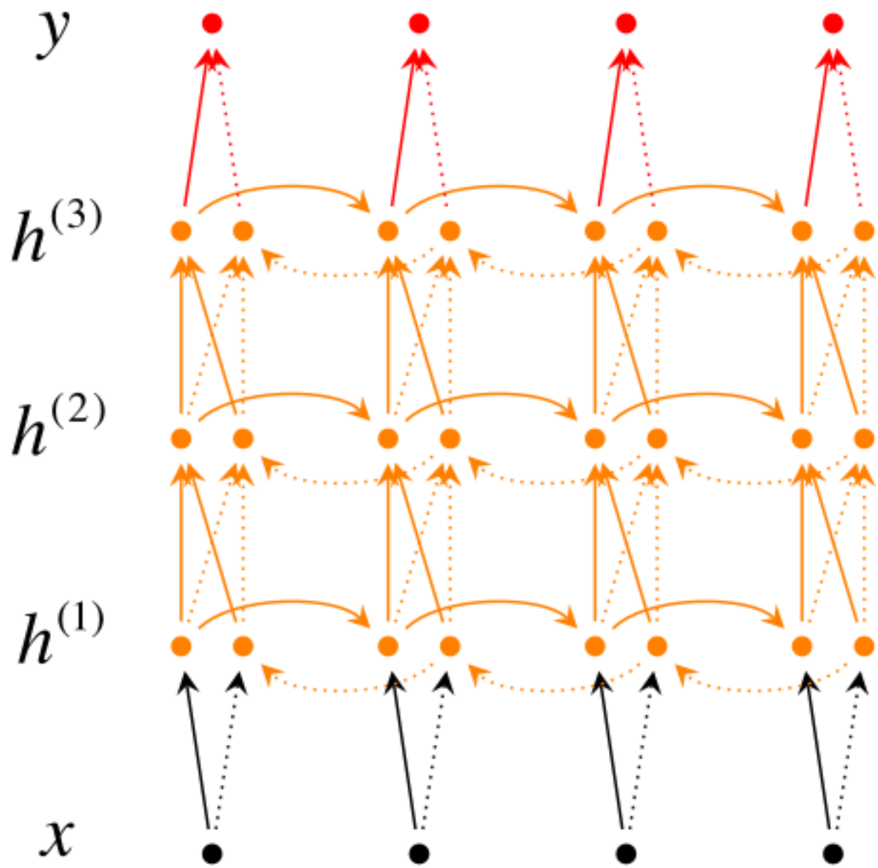
$$\overrightarrow{h}_t = f(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b})$$

$$\overleftarrow{h}_t = f(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$$y_t = g(U[\overrightarrow{h}_t;\overleftarrow{h}_t]+c)$$

$h = [\overrightarrow{h};\overleftarrow{h}]$ now represents (summarizes) the past and future around a single token.

Deep Bidirectional RNNs



$$\overrightarrow{h}_t^{(i)} = f(\overrightarrow{W}^{(i)} h_t^{(i-1)} + \overrightarrow{V}^{(i)} \overrightarrow{h}_{t-1}^{(i)} + \overrightarrow{b}^{(i)})$$

$$\overleftarrow{h}_t^{(i)} = f(\overleftarrow{W}^{(i)} h_t^{(i-1)} + \overleftarrow{V}^{(i)} \overleftarrow{h}_{t+1}^{(i)} + \overleftarrow{b}^{(i)})$$

$$y_t = g(U[\overrightarrow{h}_t^{(L)}; \overleftarrow{h}_t^{(L)}] + c)$$

Each memory layer passes an intermediate sequential representation to the next.

Data

- MPQA 1.2 corpus (Wiebe et al., 2005)
- consists of 535 news articles (11,111 sentences)
- manually labeled with DSE and ESEs at the phrase level
- Evaluation: F1

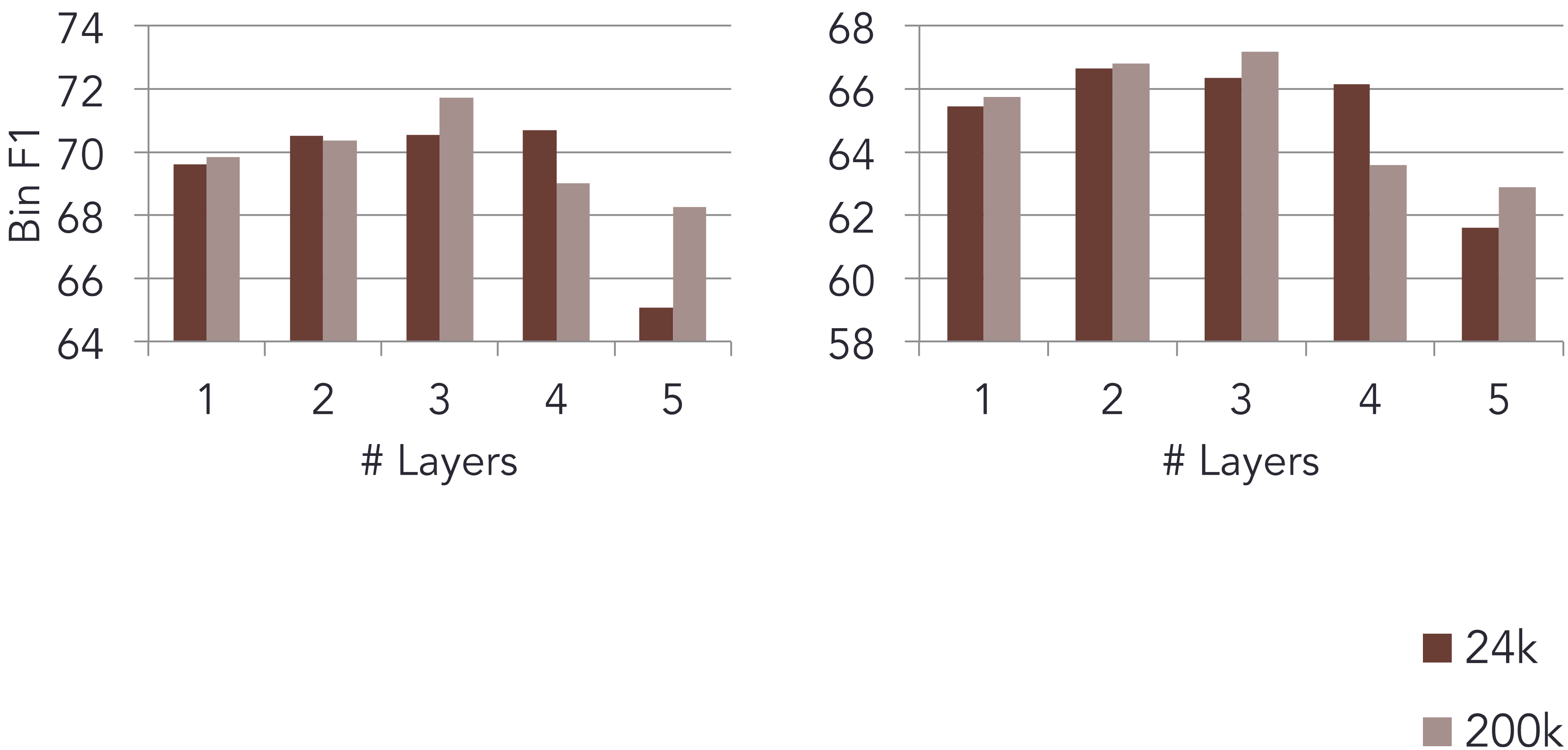
$$\text{precision} = \frac{tp}{tp + fp}$$

$$\text{recall} = \frac{tp}{tp + fn}$$

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

- Harmonic mean of precision and recall

Evaluation



74
72
70
68
66
64
Bin F1
1
2
3
4
5
Layers
68
66
64
62
60
58
1
2
3
4
5
Layers
24k
200k

Recap

- Recurrent Neural Network is one of the best deepNLP model families
- Training them is hard because of vanishing and exploding gradient problems
- They can be extended in many ways and their training improved with many tricks (more to come)
- Next week: Most important and powerful RNN extensions with LSTMs and GRUs

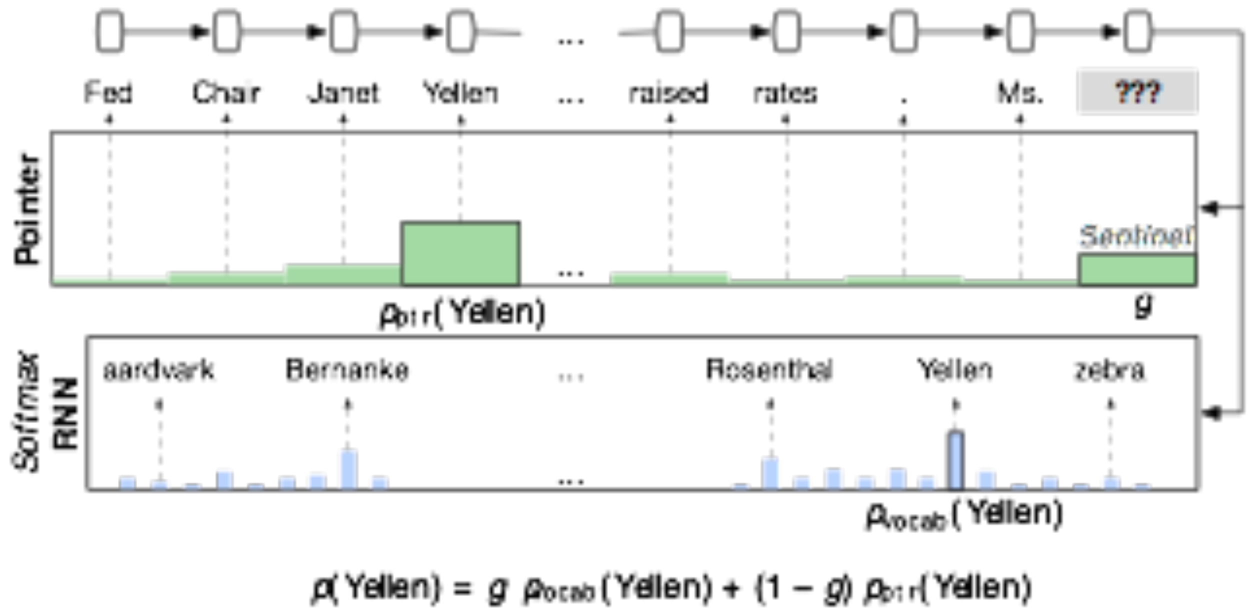
Problem with Softmax:
No Zero Shot Word Predictions

- Answers can only be predicted if they were seen during training and part of the softmax

- But it’s natural to learn new words in an active conversation and systems should be able to pick them up

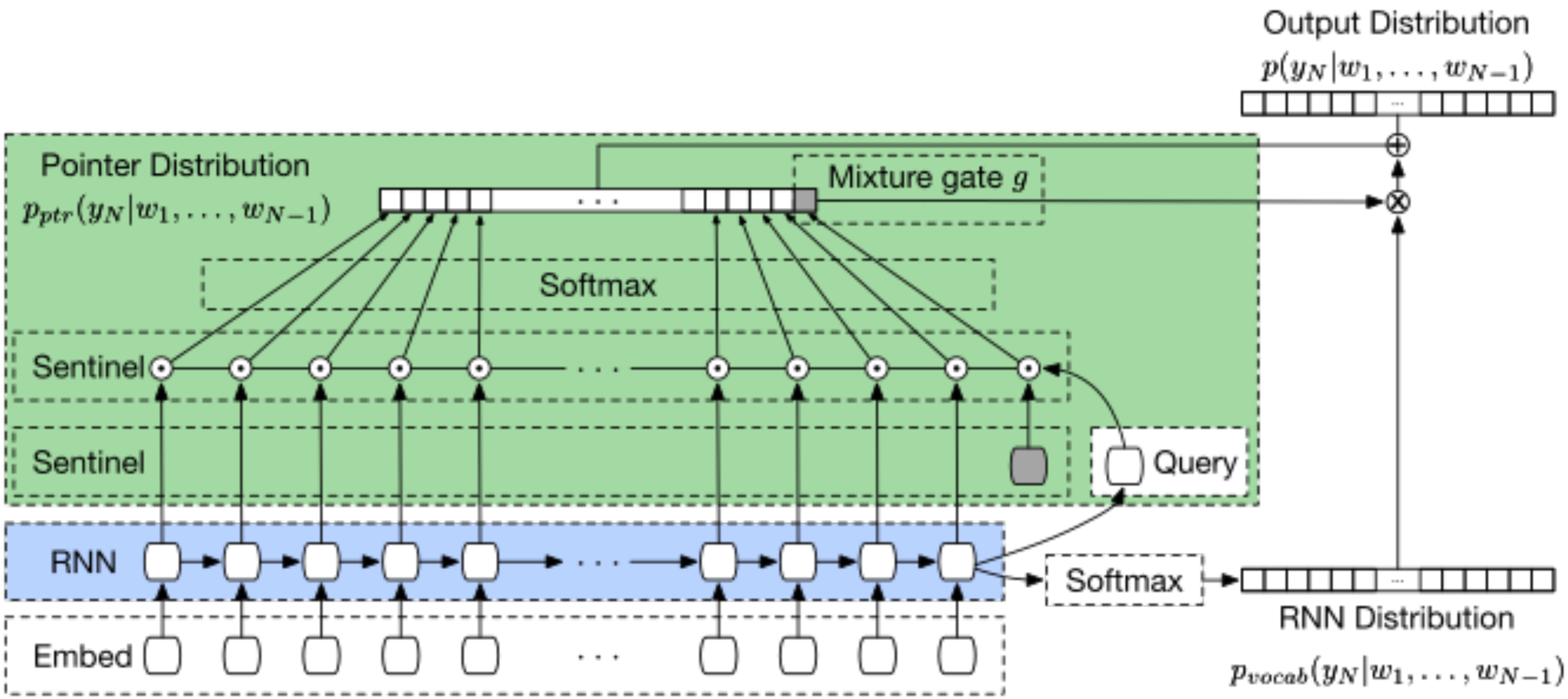
Tackling Obstacle by Predicting Unseen Words

- Idea: Mixture Model of softmax and pointers:



- Pointer Sentinel Mixture Models by Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher

Pointer-Sentinel Model - Details



$$p(y_i|x_i) = g\, p_{\text{vocab}}(y_i|x_i) + (1-g)\, p_{\text{ptr}}(y_i|x_i)$$

$$z_i = q^T h_i, \qquad p_{\text{ptr}}(w) = \sum_{i \in I(w,x)} a_i,$$

$$a = \text{softmax}(z),$$

Pointer Sentinel for Language Modeling

Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M[‡]	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M[‡]	–	125.7
Mikolov & Zweig (2012) - RNN	6M[‡]	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M[‡]	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M[‡]	–	92.0
Pascanu et al. (2013a) - Deep RNN	6M	–	107.5
Cheng et al. (2014) - Sum-Prod Net	5M[‡]	–	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	81.9 ± 0.2	79.7 ± 0.1
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	–	78.6 ± 0.1
Gal (2015) - Variational LSTM (large, untied)	66M	77.9 ± 0.3	75.2 ± 0.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	–	73.4 ± 0.0
Kim et al. (2016) - CharCNN	19M	–	78.9
Zilly et al. (2016) - Variational RHN	32M	72.8	71.3
Zoneout + Variational LSTM (medium)	20M	84.4	80.6
Pointer Sentinel-LSTM (medium)	21M	72.4	**70.9**