CS395T: Structured Models for NLP
Lecture 13: Neural Networks

Greg Durrett

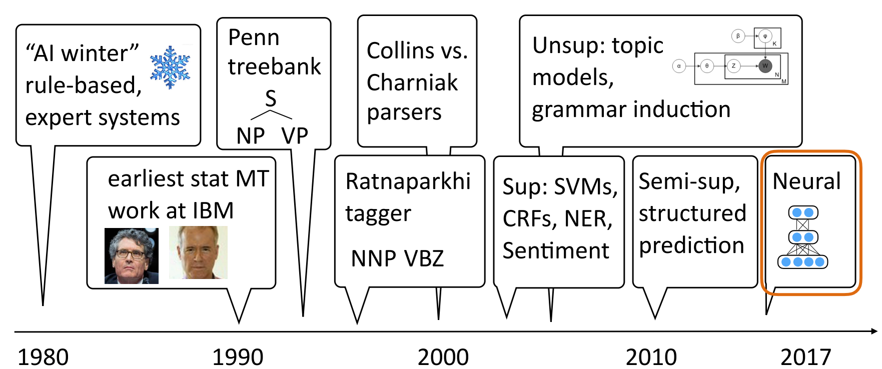
Administrivia

- Project 2 due on Tuesday
- Project 1 samples posted on website

This Lecture

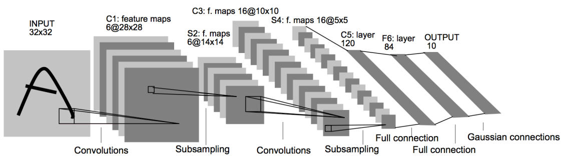
- Neural network history
- Neural network basics
- Feedforward neural networks
- Backpropagation
- Applications

A brief history of (modern) NLP

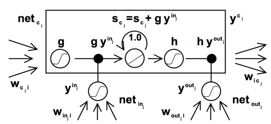


History: NN "dark ages"

- Convnets: applied to MNIST by LeCun in 1998



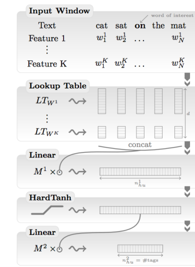
- LSTMs: Hochreiter and Schmidhuber (1997)



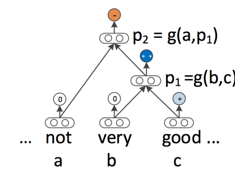
- Henderson (2003): neural shift-reduce parser, not SOTA

2008-2013: A glimmer of light…

- Collobert and Weston 2011: "NLP (almost) from scratch"
  - Feedforward neural nets induce features for sequential CRFs ("neural CRF")
  - 2008 version was marred by bad experiments, claimed SOTA but wasn't, 2011 version tied SOTA



- Krizhevskey et al. (2012): AlexNet for vision
- Socher: tree-structured RNNs
  - Started working well for sentiment in 2013, but only worked for weird tasks before that, some lackluster parsing results



2014: Stuff starts working

- Kim (2014) + Kalchbrenner et al. (2014): sentence classification / sentiment
  - Basic convnets work pretty well for NLP
- Sutskever et al., Bahdanau et al. seq2seq for neural MT
  - LSTMs actually do well at NLP problems
- Chen and Manning transition-based dependency parser
  - Feedforward neural networks for parsing
- 2015: explosion of neural nets for everything under the sun

Why didn't they work before?

- **Datasets too small**: for MT, not really better until you have 1M+ parallel sentences (and really need a lot more)
- **Optimization not well understood**: good initialization, per-feature scaling + momentum (Adagrad / Adadelta / Adam) work best out-of-the-box
  - **Regularization**: dropout was very important
  - **Computers not big enough**: can't run for enough iterations
- **Inputs**: need word representations to have the right continuous semantics
  - **Dealing with unknown words**: word pieces, use character LSTMs, … complex stuff!

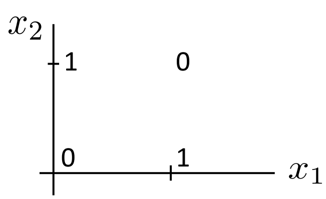
Neural Net Basics

Neural Networks

- Linear classification: $\text{argmax}_y w^\top f(x, y)$
- How can we do nonlinear classification?
- Polynomial, etc. from kernels, but these are slow!
- Kernels are neither necessary nor sufficient: not every pair of features interacts, might need to go beyonds pairs
- Instead, want to learn intermediate conjunctive features of the input

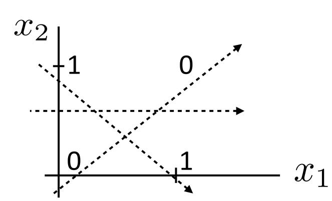
Neural Networks: XOR

- Let's see how we can use neural nets to learn a simple nonlinear function
- Inputs $x_1, \;\; x_2$
  $(\text{generally } \mathbf{x} = (x_1, \ldots, x_m))$
- Output y
  $(\text{generally } \mathbf{y} = (y_1, \ldots, y_n))$



x_1	x_2	$y = x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Neural Networks: XOR



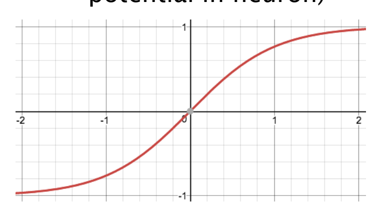
x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

$y = a_1 x_1 + a_2 x_2$ X

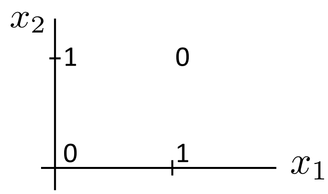
$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$ ✓

"or"

(looks like action potential in neuron)



Neural Networks: XOR



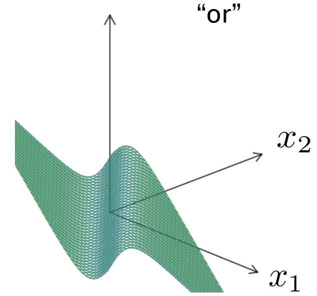
$y = a_1 x_1 + a_2 x_2$ ✗

$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$ ✓

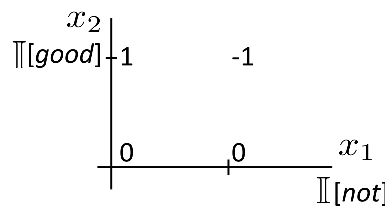
$y = -x_1 - x_2 + 2 \tanh(x_1 + x_2)$

"or"

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

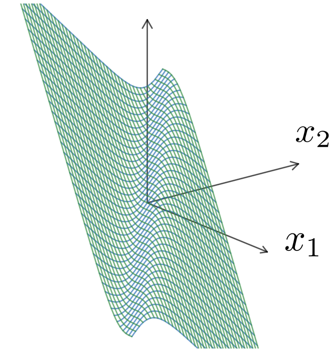


Neural Networks: XOR



$y = -2x_1 - x_2 + 2 \tanh(x_1 + x_2)$

*the movie was **not good***



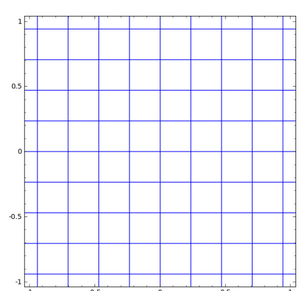
Neural Networks

(Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$)

$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

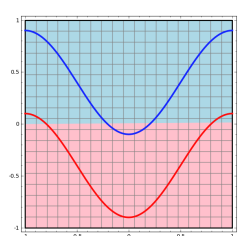
Nonlinear transformation — Warp space — Shift



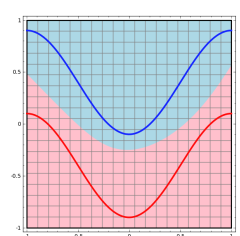
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Neural Networks

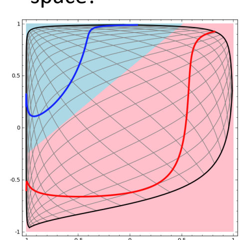
Linear classifier



Neural network



...possible because we transformed the space!

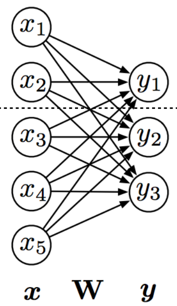


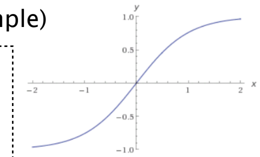
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Neural Networks

(this was our neural net from the XOR example)

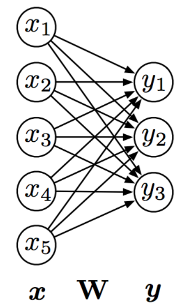
$y_1 = g(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$



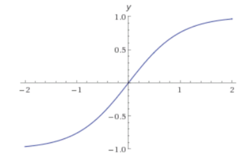


Adopted from Chris Dyer

Deep Neural Networks



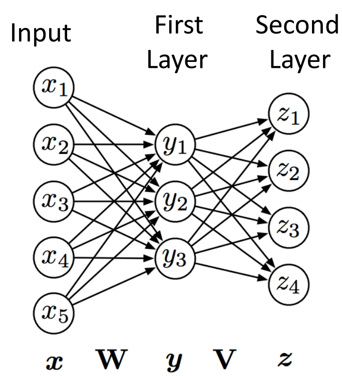
$y_1 = g(\mathbf{w}_1 \cdot \mathbf{x} + b_1)$



$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$

Adopted from Chris Dyer

Deep Neural Networks



$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$

$\mathbf{z} = g(\mathbf{V}\mathbf{y} + \mathbf{c})$

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$

"Feedforward": computation "feeds forward" (not recurrent)

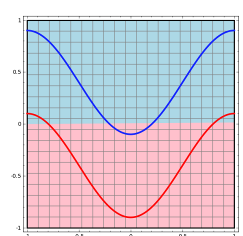
Check: what happens if no nonlinearity?
More powerful than basic linear models?

$\mathbf{z} = \mathbf{V}(\mathbf{W}\mathbf{x} + \mathbf{b}) + \mathbf{c}$

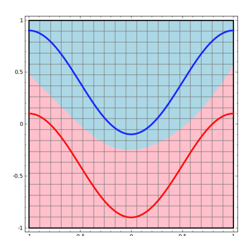
Adopted from Chris Dyer

Deep Neural Networks

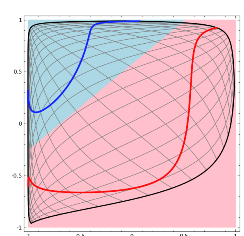
Linear classifier



Neural network

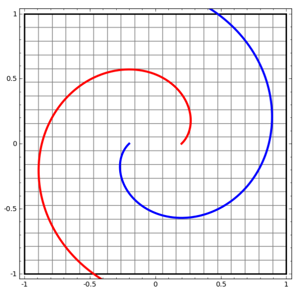


...possible because we transformed the space!



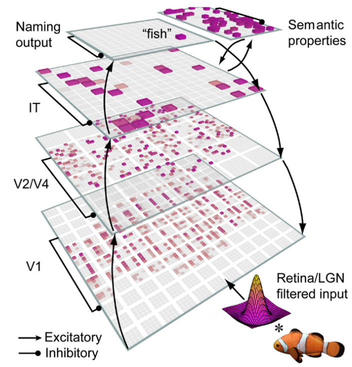
Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Neural Networks



Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Deep Neural Networks



- Using multiple layers of processing to induce deep representations parallels visual processing in the brain

From O'Reilly et al. (2013)

Feedforward Networks, Backpropagation

Logistic Regression with NNs

$$P(y|\mathbf{x}) = \frac{\exp(w^\top f(\mathbf{x}, y))}{\sum_{y'} \exp(w^\top f(\mathbf{x}, y'))}$$

- Single scalar probability

$$P(y|\mathbf{x}) = \text{softmax}_y(w^\top f(\mathbf{x}, y))$$

- softmax_y: score vector -> prob of y

$$P(y|\mathbf{x}) = \text{softmax}_y(w_y^\top \underbrace{g(Vf(\mathbf{x}))})$$

Hidden representation **z**, can see this as "induced features"

- Feature function no longer looks at label — same shared processing for each label.

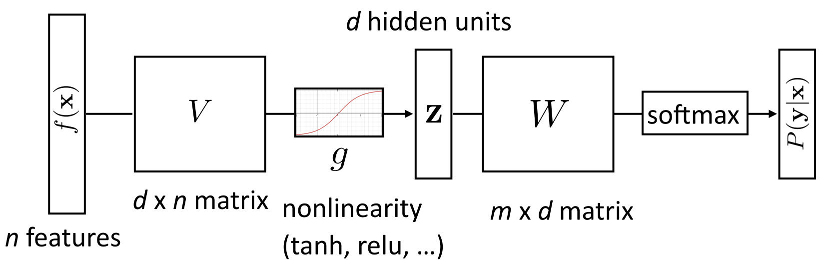
$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$

- softmax: score vector -> probability vector

- Assumes that the labels *y* are indexed and associated with coordinates in a vector space

Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



Training Neural Networks

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$

- Maximize log likelihood of training data

$$\log P(y = i^*|\mathbf{x}) = \log\left(\mathrm{softmax}(Wg(Vf(\mathbf{x}))) \cdot e_{i^*}\right)$$

- i^*: index of the gold label
- e_i: 1 in the ith row, zero elsewhere. Dot by this = select ith index

$$\mathcal{L}(\mathbf{x}, i^*) = Wg(Vf(\mathbf{x})) \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wg(Vf(\mathbf{x})) \cdot e_j)$$

Computing Gradients

$$\mathcal{L}(\mathbf{x}, i^*) = Wg(Vf(\mathbf{x})) \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(Wg(Vf(\mathbf{x})) \cdot e_j)$$

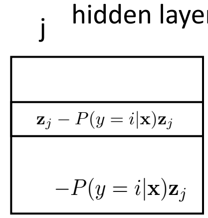
$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(W\mathbf{z} \cdot e_j)$$

$\mathbf{z} = g(Vf(\mathbf{x}))$

Activations at hidden layer

- Gradient with respect to *W*

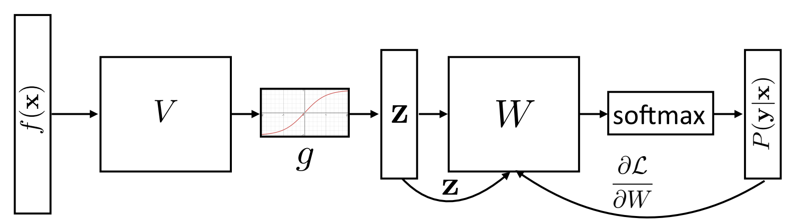
$$\frac{\partial}{\partial W_{ij}} \mathcal{L}(\mathbf{x}, i^*) = \begin{cases} \mathbf{z}_j - P(y = i|\mathbf{x})\mathbf{z}_j & \text{if i = i*} \\ -P(y = i|\mathbf{x})\mathbf{z}_j & \text{otherwise} \end{cases}$$



- Looks like logistic regression with ***z*** as the features!

Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



Computing Gradients: Backpropagation

$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(W\mathbf{z} \cdot e_j)$$

$\mathbf{z} = g(Vf(\mathbf{x}))$
Activations at hidden layer

- Gradient with respect to *V*: apply the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{V_{ij}}$$

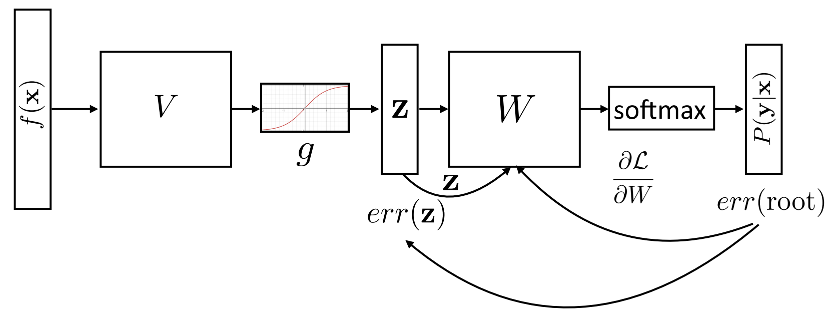
$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = W_{i^*} - \sum_j P(y = j|\mathbf{x}) W_j$$

vector vector

- weights(gold) - E[weights(guess)], like LR with weights and features flipped!
- Or: $err(\text{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$ dim = m $\quad \frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = err(\mathbf{z}) = W^\top err(\text{root})$ dim = d

Backpropagation: Picture

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



Computing Gradients: Backpropagation

$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_{j=1}^{m} \exp(W\mathbf{z} \cdot e_j)$$

$\mathbf{z} = g(Vf(\mathbf{x}))$
Activations at hidden layer

- Gradient with respect to *V*: apply the chain rule

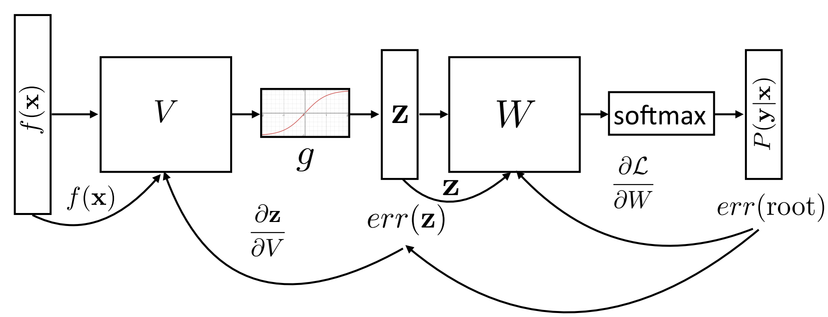
$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{V_{ij}}$$

$$\frac{\partial \mathbf{z}}{V_{ij}} = \frac{\partial g(\mathbf{a})}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial V_{ij}} \qquad \mathbf{a} = Vf(\mathbf{x})$$

- First term: gradient of nonlinear activation function at ***a*** (depends on current value)
- Second term: gradient of linear function
- Straightforward computation once we have *err*(**z**)

Backpropagation: Picture

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$



Backpropagation

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$

- Step 1: compute $err(\text{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$ (vector)
- Step 2: compute derivatives of *W* using *err*(root) (matrix)
- Step 3: compute $\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = err(\mathbf{z}) = W^\top err(\text{root})$ (vector)
- Step 4: compute derivatives of *V* using *err*(**z**) (matrix)
- Step 5+: continue backpropagation (compute err(*f*(**x**)) if necessary…)

Backpropagation: Takeaways

- Gradients of output weights *W* are easy to compute — looks like logistic regression with hidden layer ***z*** as feature vector
- Can compute derivative of loss with respect to ***z*** to form an "error signal" for backpropagation
- Easy to update parameters based on "error signal" from next layer, keep pushing error signal back as backpropagation
- Need to remember the values from the forward computation

Applications

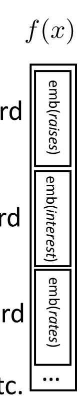
NLP with Feedforward Networks

- Part-of-speech tagging with FFNNs

??

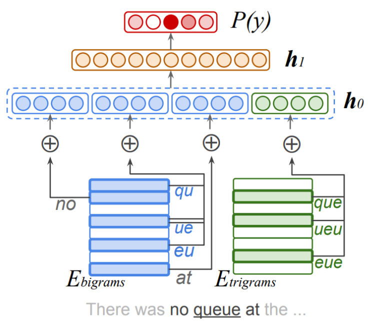
Fed raises ***interest*** *rates in order to …*

- Word embeddings for each word form input
- ~1000 features here — smaller feature vector than in sparse models, but every feature fires on every example
- Weight matrix learns position-dependent processing of the words



Botha et al. (2017)

NLP with Feedforward Networks



- Hidden layer mixes these different signals and learns feature conjunctions

Botha et al. (2017)

NLP with Feedforward Networks

- Multilingual tagging results:

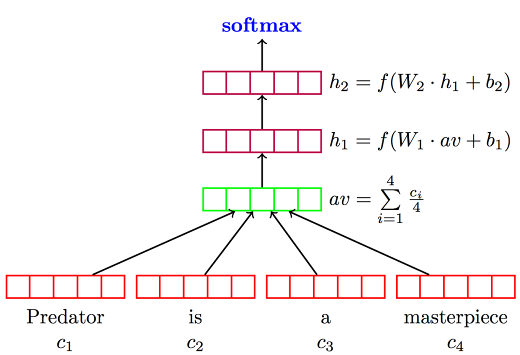
Model	Acc.	Wts.	MB	Ops.
Gillick et al. (2016)	95.06	900k	-	6.63m
Small FF	94.76	241k	0.6	0.27m
+Clusters	95.56	261k	1.0	0.31m
$\frac{1}{2}$ Dim.	95.39	143k	0.7	0.18m

- Gillick used LSTMs; this is smaller, faster, and better

Botha et al. (2017)

Sentiment Analysis

- Deep Averaging Networks: feedforward neural network on average of word embeddings from input



Iyyer et al. (2015)

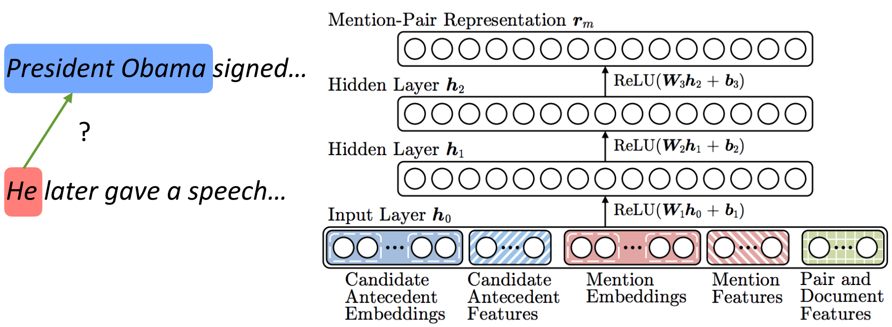
Sentiment Analysis

	Model	RT	SST fine	SST bin	IMDB	Time (s)	
	DAN-ROOT	—	46.9	85.7	—	**31**	
	DAN-RAND	77.3	45.4	83.2	88.8	136	
	DAN	80.3	47.7	86.3	89.4	136	
Bag-of-words	NBOW-RAND	76.2	42.3	81.4	88.9	91	
	NBOW	79.0	43.6	83.6	89.0	91	
	BiNB	—	41.9	83.1	—	—	
	NBSVM-bi	79.4	—	—	91.2	—	Wang and Manning (2012)
Tree RNNs / CNNS / LSTMS	RecNN*	77.7	43.2	82.4	—	—	
	RecNTN*	—	45.7	85.4	—	—	
	DRecNN	—	49.8	86.6	—	431	
	TreeLSTM	—	**50.6**	86.9	—	—	
	DCNN*	—	48.5	86.9	89.4	—	
	PVEC*	—	48.7	87.8	**92.6**	—	
	CNN-MC	**81.1**	47.4	**88.1**	—	2,452	Kim (2014)
	WRRBM*	—	—	—	89.2	—	

Iyyer et al. (2015)

Coreference Resolution

- Feedforward networks identify coreference arcs



Clark and Manning (2015), Wiseman et al. (2015)

Next Time

- How to implement neural networks for NLP
  - Tensorflow
  - Practical training techniques
- Word representations / word vectors
- word2vec, GloVe