

NLP from (almost) Scratch

Bhuvan Venkatesh, Sarah
Schieferstein (bvenkat2,
schfrst2)

Introduction

Motivation

- Models for NLP have become too specific
- We engineer features and hand pick models so that we boost the accuracy on certain tasks
- We don't want to create a general AI, but we also want machine learning models to share parameters

Importance

- Just to reiterate - this paper is seminal
- They were using Neural Nets in 2011, way before they were cool
- They also unearthed some challenges about the neural nets and the amount of data needed.
- The coolest part is that it doesn't need to be labeled, cheapening the entire training process

Existing Benchmarks

Tasks

- The paper focuses on 4 similar but unrelated tasks
 - POS - Part of Speech tagging
 - CHUNK - Chunking
 - NER - Named Entity Recognition
 - SRL - Semantic Role Labeling
- For the metrics for POS, they used Accuracy. For everything else, they use F1 score.

Traditional Systems

- Traditional Systems have pretty good accuracy. Most of them are ensemble models that incorporate a bunch of classical NLP features and use a standard algorithm like SVM [3] or Markov Model [2]
- Some of the models use bidirectional models like a BiLSTM to capture context from both ends

System	Accuracy
Shen et al. (2007)	97.33%
Toutanova et al. (2003)	97.24%
Giménez and Màrquez (2004)	97.16%

(a) POS

System	F1
Shen and Sarkar (2005)	95.23%
Sha and Pereira (2003)	94.29%
Kudo and Matsumoto (2001)	93.91%

(b) CHUNK

System	F1
Ando and Zhang (2005)	89.31%
Florian et al. (2003)	88.76%
Kudo and Matsumoto (2001)	88.31%

(c) NER

System	F1
Koomen et al. (2005)	77.92%
Pradhan et al. (2005)	77.30%
Haghighi et al. (2005)	77.04%

(d) SRL

Notes

- Only chose systems that didn't dabble with external data. Meaning they didn't train with extra data or didn't introduced additional features (i.e. POS for NER task) using either another ML algorithm or hand drawn annotating.
- The features ended up being heavily engineered for the task at hand to achieve hair-pin accuracy

Network Approach

Overview

- We are going to preprocess features as little as possible and feed a representation into the neural net.
- If we want to put any discrete features like POS, capitalization, or stems, we can concatenate a one-hot encoded vector turning the feature “on”

Windowed vs Convolutional

- The authors described two flavors of neural networks. One considers a window of words with word paddings
- The other is a sentence approach that takes a convolutional filter of a certain size and applies it before performing the neural model

Input Window

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

word of interest

Lookup Table

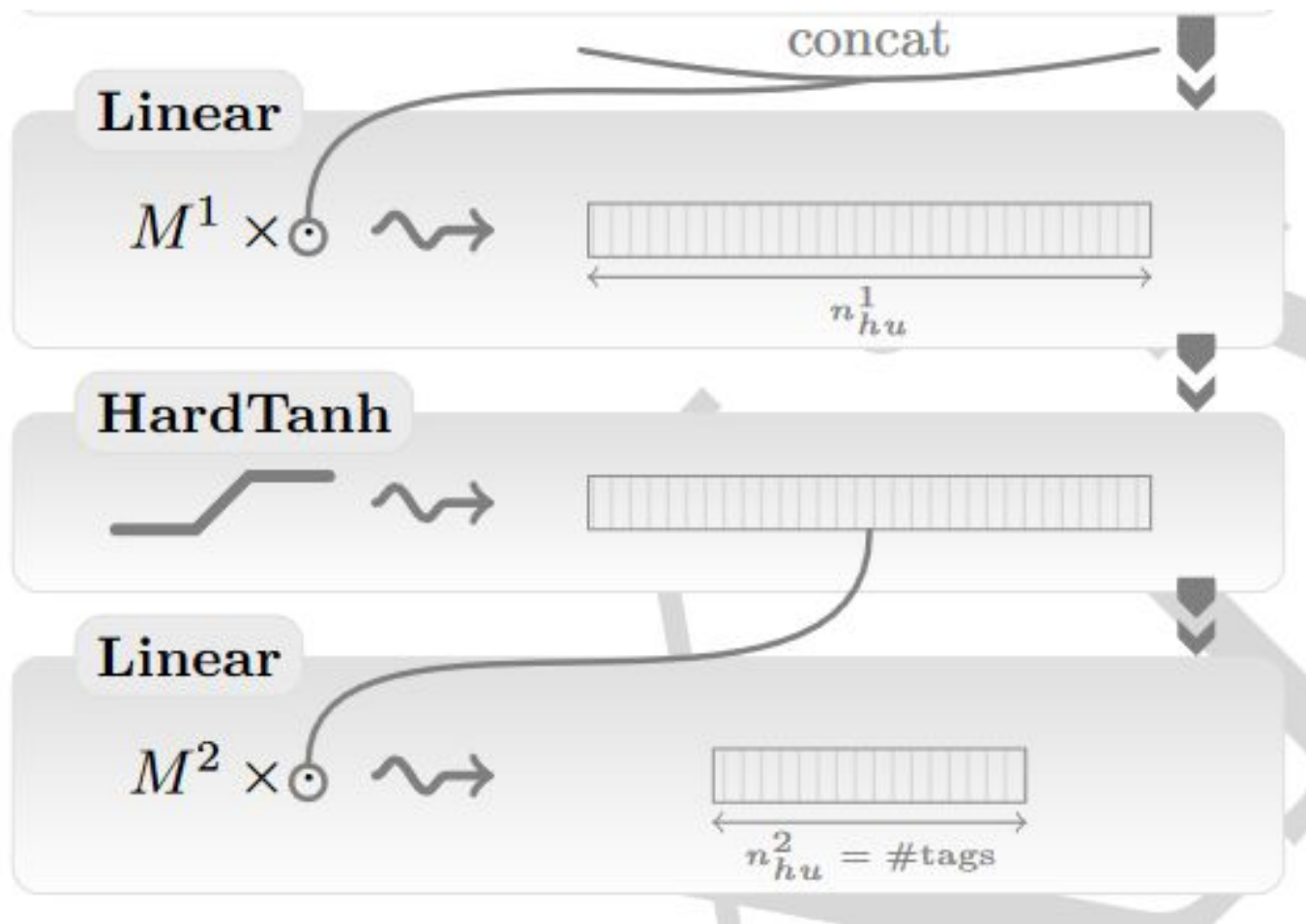
LT_{W^1} \rightsquigarrow

⋮

LT_{W^K} \rightsquigarrow



concat



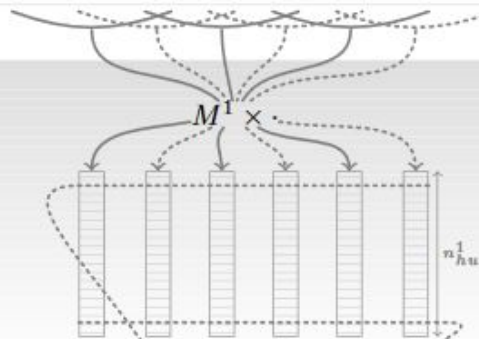
Input Sentence

Text		The	cat	sat	on	the	mat	
Feature 1	<i>Padding</i>	w_1^1	w_2^1	...			w_N^1	<i>Padding</i>
⋮								
Feature K		w_1^K	w_2^K	...			w_N^K	

Lookup Table



Convolution



Max Over Time



Additional Considerations

- The max layer is a pooling layer with a variable sized window so that there are a constant number of features (if not some padding is added)
- For all tasks but POS, they will use IOBES encoding which mean: Inside, Other, Beginning, Ending, Single. This is so that each word turns into a classification task which is easily measured through F1
- For some tasks, they put in stemming or capitalization

Embeddings

- For this stage of the algorithm we are going to learn the embeddings along with the neural net. We initialize the word embeddings randomly and train using back prop.

$$\frac{\partial C}{\partial \langle W \rangle_i^1} = \sum_{\{1 \leq t \leq T / [w]_t = i\}} \left\langle \frac{\partial C}{\partial f_{\theta}^l} \right\rangle_i^1$$

- Spoiler Alert: This will tend not to be a good idea because the Nets would like to have a fixed representation

Two Gradients, both alike in dignity

- Windowed Gradient - Use cross entropy with a softmax, pushing down all non relevant probabilities.
- They mention that this is a problem because cross entropy is good with low covariant features but tags usually depend on the context surrounding them

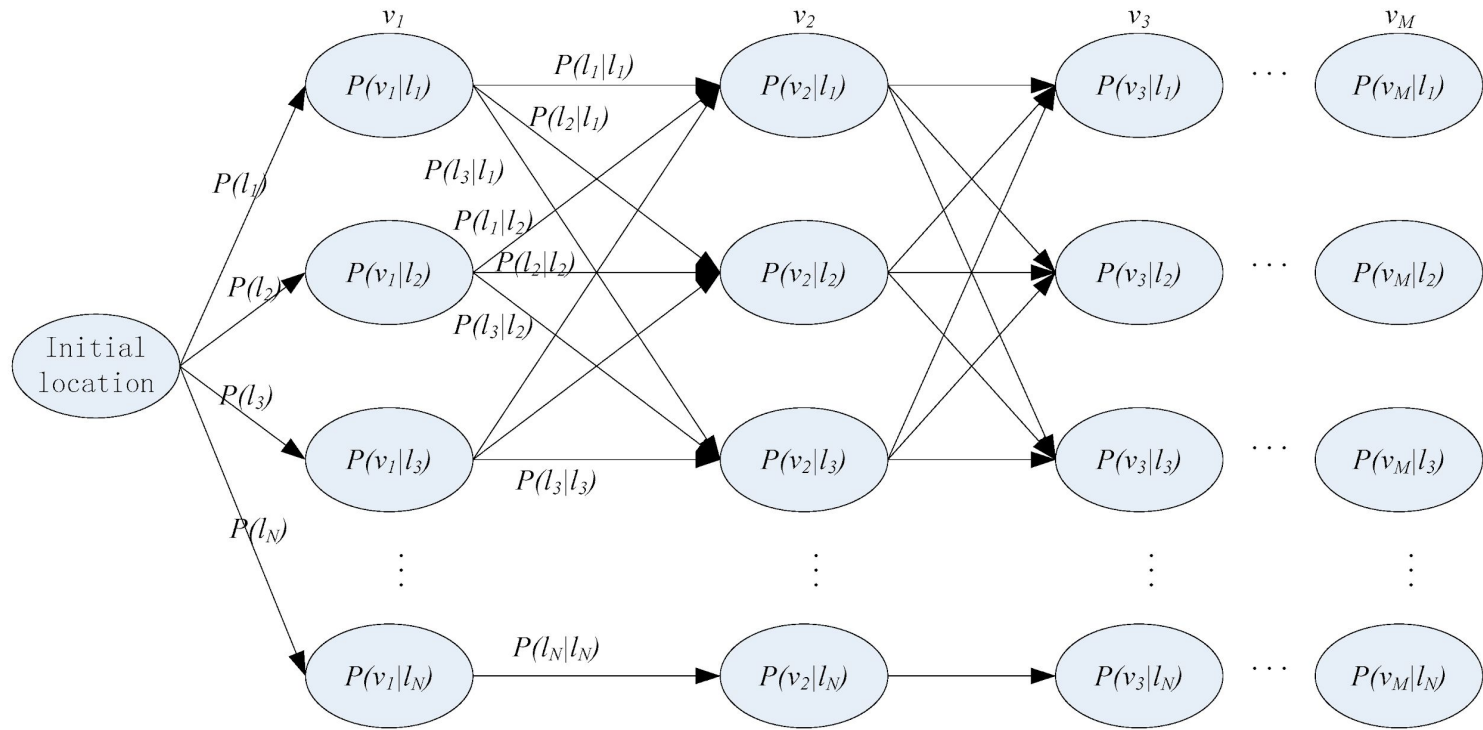
Gradient 1

$$p(i|x, \theta) = \frac{e^{[f_\theta]_i}}{\sum_j e^{[f_\theta]_j}}.$$

$$\text{logadd } z_i = \log\left(\sum_i e^{z_i}\right),$$

What of Capulet? (Sentence Loss)

- Sentence level gradient - We are going to use a trellis to illustrate what this does
- We introduce new transition parameters A that give the probability of going from tag 1 to tag 2 in a sentence. We will train it with the rest of the model.
- We take all possible paths through the tags and words and softmax the probability of all paths with the actual path the word takes



Notation

- $s([x]_1^T, [i]_1^T, \tilde{\theta})$ - The probability of sentence x starting at element number 1 has a particular tag sequence $[i]$ starting at tag 1
- $[A]_{[i]_{t-1}, [i]_t} + [f_{\theta}]_{[i]_t, t}$ - The A term is the probability that tag $[i]_{t-1}$ turns into $[i]_t$ at time t . The f term is the probability that the word takes tag $[i]_t$ at time t . (Time being number word in the sentence)
- Logadd is logadd previously

$$s([\mathbf{x}]_1^T, [i]_1^T, \tilde{\boldsymbol{\theta}}) = \sum_{t=1}^T \left([A]_{[i]_{t-1}, [i]_t} + [f_{\boldsymbol{\theta}}]_{[i]_t, t} \right)$$

$$\log p([\mathbf{y}]_1^T \mid [\mathbf{x}]_1^T, \tilde{\boldsymbol{\theta}}) = s([\mathbf{x}]_1^T, [\mathbf{y}]_1^T, \tilde{\boldsymbol{\theta}}) - \log_{\forall [j]_1^T} \text{add} s([\mathbf{x}]_1^T, [j]_1^T, \tilde{\boldsymbol{\theta}}).$$

But wait, Loss Function?

- The unoptimized loss function becomes exponential to compute and the gradient as such.
- The way they optimize it is using the ring properties of the loss function
- The same effect can be achieved by running a modified Viterbi algorithm that stores the cumulative sum and the current path probabilities

Inference?

- At inference time for the windowed approach, just take the argmax of the output layer in the neural network to find the class of the word in the middle of the window
- For the other approach, use the neural net to get a list of tag probabilities at a particular time. Then use the Viterbi algorithm to predict the most likely sequence of labelings.

Note: Conditional Random Fields

- Similar but there are a few differences, one being that our path probability function is **not normalized** (which in training can help avoid the label-bias problem where a sequence of states may be less likely than a state-to-state probability) [5]
- But in a different sense, this is the same as training a CRF except now we are training a non-linear model to get the output activations instead of a linear one.

Training?

- Train with stochastic gradient descent, nothing fancy
- For the windowed approach, compute the gradient in the window; the sentence approach, the gradient in the whole sentence
- Hyperparameters chosen by validation, learning rate does not change over time though so convergence is not guaranteed.

Hyperparameters

Task	Window/Conv. size	Word dim.	Caps dim.	Hidden units	Learning rate
POS	$d_{win} = 5$	$d^0 = 50$	$d^1 = 5$	$n_{hu}^1 = 300$	$\lambda = 0.01$
CHUNK	”	”	”	”	”
NER	”	”	”	”	”
SRL	”	”	”	$n_{hu}^1 = 300$ $n_{hu}^2 = 500$	”

Results

Approach	POS (PWA)	Chunking (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99

Performance

- Not too bad for out of the box performance with minimal tuning. It matches with the baseline fairly well within single 1-7% percent differences of specialized models
- Very low learning parameter, so it takes a long time to learn
- Small window so results are expected for long term dependencies

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
PERSUADE FAW	THICKETS SAVARY	DECADENT DIVO	WIDESCREEN ANTICA	ODD ANCHIETA	PPA UDDIN
BLACKSTOCK GIORGI	SYMPATHETIC JFK	VERUS OXIDE	SHABBY AWE	EMIGRATION MARKING	BIOLOGICALLY KAYAK
SHAHEED RUMELIA	KHWARAZM STATIONERY	URBINA EPOS	THUD OCCUPANT	HEUER SAMBHAJI	MCLARENS GLADWIN
PLANUM GOA'ULD	ILIAS GsNUMBER	EGLINTON EDGING	REVISED LEAVENED	WORSHIPPERS RITSUKO	CENTRALLY INDONESIA
COLLATION BACHA	OPERATOR W.J.	FRG NAMSOS	PANDIONIDAE SHIRT	LIFELESS MAHAN	MONEO NILGIRIS

Better Word Embeddings with Unlabeled Data

How to get better word embeddings?

- Since the lookup table has many parameters (**ddim** x **|Dictionary|**) = (50 x 100,000) we need more data
- Use massive amounts of unlabeled data to make a window-approach language model

Datasets

- Entire English wikipedia (631 million words) tokenized with Penn Treebank script
 - Regular WSJ dictionary of 100k most frequent words
 - OOV replaced with RARE
- Reuters RCV1 (221 million words)
 - Regular WSJ dictionary of 100k most frequent word + 30k most frequent words from this dataset
 - Perhaps adding more unlabeled data will make our model better?

If it's unlabeled, how does it train?

We want to convince the model to produce LEGAL phrases.

Legal = window seen in training data

Illegal = window not seen in training data

We don't need labels for this.

Which training criterion?

Cross-entropy

$$p(i|x, \theta) = \frac{e^{[f_\theta]_i}}{\sum_j e^{[f_\theta]_j}} \quad H(p, q) = - \sum_x p(x) \log q(x).$$

- Used in our supervised models
- Normalization term is costly
- Favors frequent phrases too much
- Weights **rare** and **legal** phrases less
- We want to learn rare syntax as well to train word embeddings, though!

Pairwise ranking

$$\theta \mapsto \sum_{x \in \mathcal{X}} \sum_{w \in \mathcal{D}} \max \{ 0, 1 - f_\theta(x) + f_\theta(x^{(w)}) \}$$

- Only wants to rank one in pair as better
- Does not favor the ‘best’ ranking, so **rare** legal phrases are favored as much as **frequent** legal phrases
- Useful for word embeddings because all legal syntax is learned

Pairwise Ranking Criterion

$$\theta \mapsto \sum_{x \in \mathcal{X}} \sum_{w \in \mathcal{D}} \max \left\{ 0, 1 - f_{\theta}(x) + f_{\theta}(x^{(w)}) \right\}$$

- Attempts to make **legal** score ≥ 1 greater than ANY **illegal** score
 - \mathcal{X} : All windows in training data
 - \mathcal{D} : All words in dictionary
 - $x^{(w)}$: window with center word replaced with w . An **illegal** phrase
- Because it is pairwise and ranked, all contexts are learned and treated equally despite frequency unlike in cross-entropy

Training the model

- Use SGD to minimize the criterion
- Computers were slow and it took **weeks** to train models this large

Hyperparameter choice through breeding

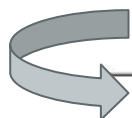
- Since it was so slow in 2011, we use the biological idea of “breeding” instead of a full grid search

Breeding process given k processors and hyper-parameters $\lambda, d, n_{hu}^h, d_{win}$

1. Train k models over several days with k different parameter sets
2. Select the best models based on validation set tests with lowest pairwise ranking loss (error)
3. Choose k new parameter sets that are permutations that are close to the best candidates
4. Initialize each new network with earlier embeddings and use a larger dictionary size each time

Word embedding results

Both models used $d_{win} = 11$, $n_{hu}^h = 100$. All other parameters matched the labeled networks.



FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
454	1973	6909	11724	29869	87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

France ~ Austria!

Wikipedia LM's shortest euclidean distance from word embeddings of various frequencies. More frequent to less.

Supervised models with these embeddings

- ‘Semi-supervised’
- Initialize lookup tables with embeddings from either language model
- Separate long embedding training from fast supervised taggers

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	77.92
NN+WLL	96.31	89.13	79.53	55.40
NN+SLL	96.37	90.33	81.47	70.99
NN+WLL+LM1	97.05	91.91	85.68	58.18
NN+SLL+LM1	97.10	93.65	87.58	73.84
NN+WLL+LM2	97.14	92.04	86.96	58.34
NN+SLL+LM2	97.20	93.63	88.67	74.15

Performance **increases** with pre-trained embeddings!

Still **not** better than feature-engineered benchmarks

Multitask Learning

A Single Model

- Now that our models behave well separately, we wish to combine them into one.
- Input = text with several features/labels, output = POS, CHUNK, NER, SRL

How do we do this? Will it boost performance as the tasks learn from each other?

Method 1: Joint decoding

- Don't train tasks together at all.
- Combine all of the models' predictions and decode their results in the same probability space
- This method ignores any inter-task dependencies; joint training is usually superior

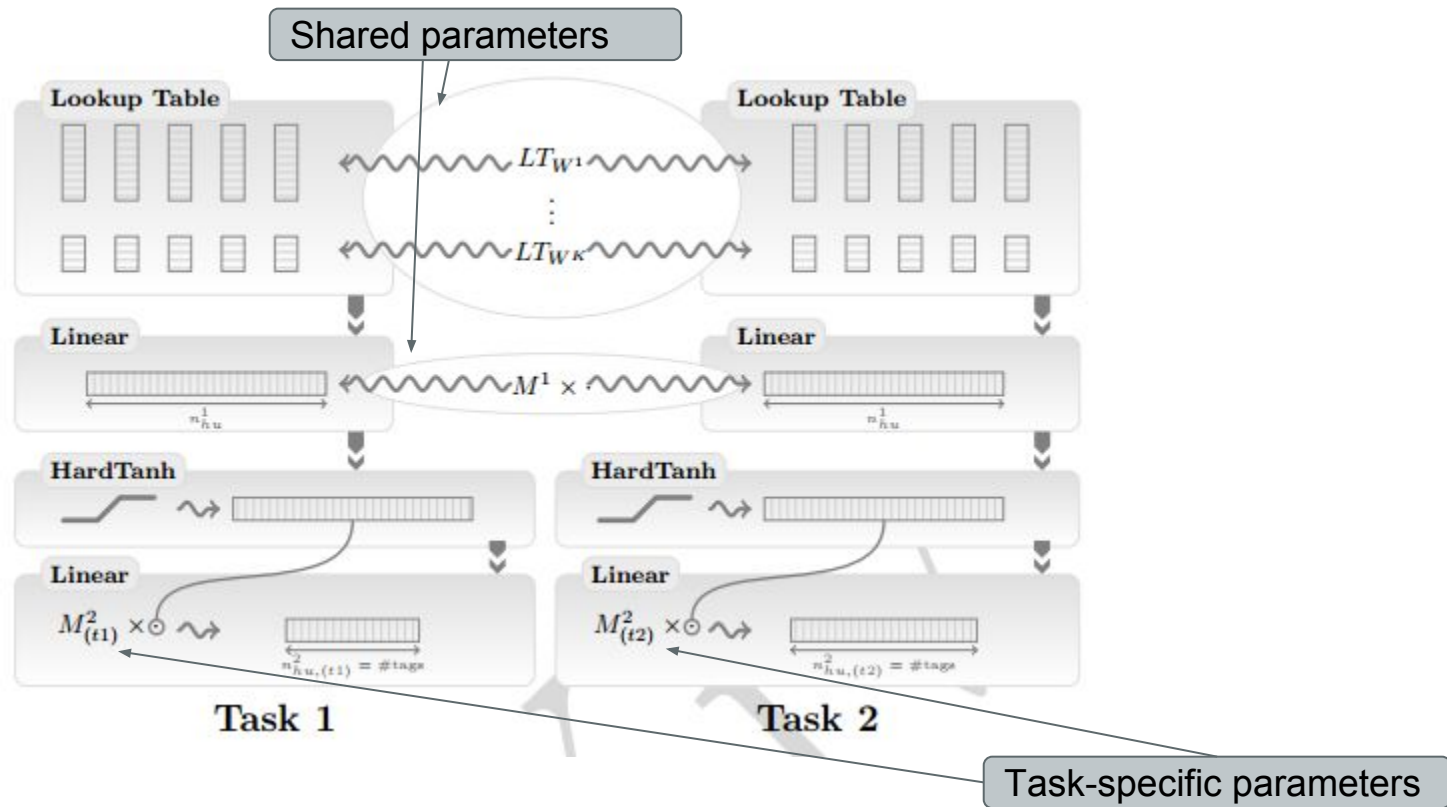
Method 2: Joint training

- Helps discover common internal representations across tasks
- Simplest method is training tasks simultaneously by sharing certain parameters
- Some parameters are denoted as task-specific and not shared

How did this paper jointly train?

- Shared parameters: lookup table, first hidden layer OR convolutional layer
- The last layer is not shared and is task-specific
- Average loss is minimized across all tasks with SGD
 - At each iteration, pick a random example from a random task
 - Apply the backpropagation results to the respective model's task-specific parameters AND its shared parameters

General Example of MTL with NN



Models created:

1. POS, CHUNK, NER trained jointly with window network. The first linear layer parameters were shared. The lookup table parameters were shared.
2. POS, CHUNK, NER, SRL trained jointly with sentence network. Convolutional layer parameters were shared. The lookup table parameters were shared.

Results

Doesn't increase performance much; language model word embeddings helped more

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL (F1)
Benchmark Systems	97.24	94.29	89.31	-
	<i>Window Approach</i>			
NN+SLL+LM2	97.20	93.63	88.67	-
NN+SLL+LM2+MTL	97.22	94.10	88.62	-
	<i>Sentence Approach</i>			
NN+SLL+LM2	97.12	93.37	88.78	74.15
NN+SLL+LM2+MTL	97.22	93.75	88.27	74.29

Good news: we have a model that takes input and outputs labels for 3+ tasks, and it is nearly as accurate as the much slower and complex benchmarks.

Task Specific Optimizations

(almost) & the temptation

- We've been doing NLP *from scratch* this whole time
- What happens if we utilize a priori knowledge and feature engineer on these neural networks? We are already close to state of the art without them...

Suffix Features

- Suffixes can predict syntactic function (-ly for adverbs, -ed for verbs...)
- In the POS task: add discrete word features in the form of a suffix dictionary
 - Use the last two characters of every word

Gazetteers

- Gazetteers: a large dictionary of well known named entities
 - 4 categories: locations, names, orgs, misc. = 4 features
- In the NER task: if a chunk is in the gazetteer, the chunk's words in the respective feature/category is turned to 'on'
- Vastly improves NER, likely due to chunk information (our language model does not consider chunks)

Cascading

- Use features obtained from previous tasks
- For CHUNK and NER: add discrete word features that represent POS tag of each word

Approach	POS (PWA)	CHUNK (F1)	NER (F1)	SRL
Benchmark Systems	97.24	94.29	89.31	77.92
NN+SLL+LM2	97.20	93.63	88.67	74.15
NN+SLL+LM2+Suffix2	97.29	-	-	-
NN+SLL+LM2+Gazetteer	-	-	89.59	-
NN+SLL+LM2+POS	-	94.32	88.67	-
NN+SLL+LM2+CHUNK	-	-	-	74.72

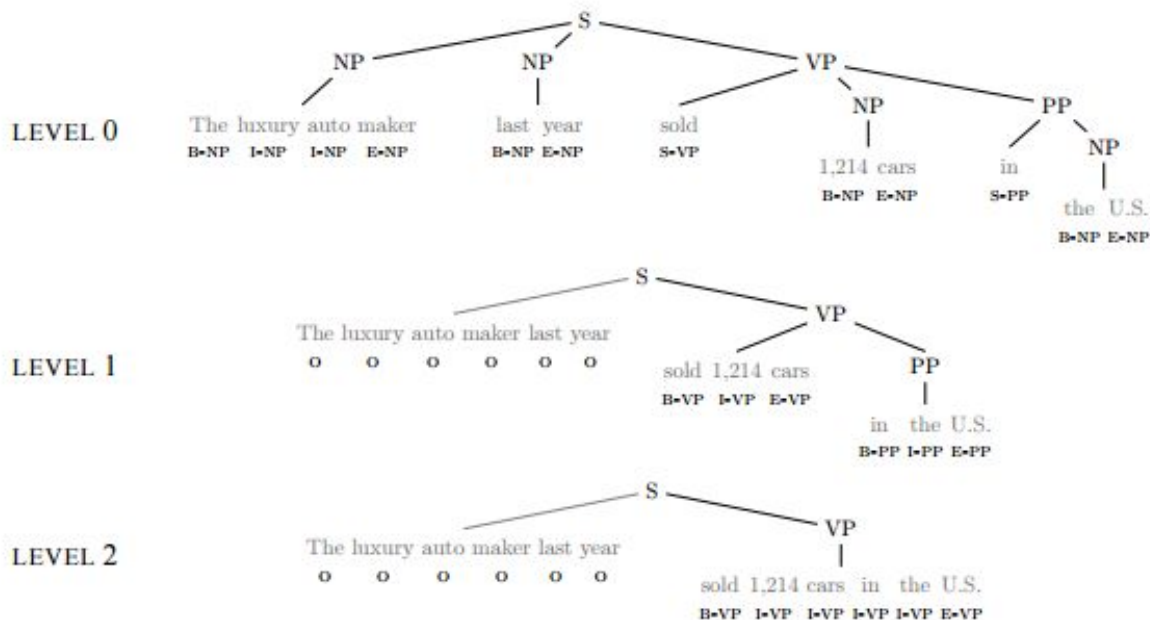
Ensembles

- Combine outputs of multiple classifiers (with random initial parameters)
- Done for POS, CHUNK, NER
- Voting ensemble: take majority vote, one vote is the tag each model estimated
- Joined ensemble: combine model outputs (NOT tags, the feature vectors) with another linear layer, then finally feed to SLL. This doesn't perform as well as voting.

Approach		POS (PWA)	CHUNK (F1)	NER (F1)
Benchmark Systems		97.24	94.29	89.31
NN+SLL+LM2+POS	worst	97.29	93.99	89.35
NN+SLL+LM2+POS	mean	97.31	94.17	89.65
NN+SLL+LM2+POS	best	97.35	94.32	89.86
NN+SLL+LM2+POS	voting ensemble	97.37	94.34	89.70
NN+SLL+LM2+POS	joined ensemble	97.30	94.35	89.67

Parsing

- In SRL task: feed in a parse tree and its successive 'levels' (i.e. collapsing terminals upward)



Approach	SRL	
	(valid)	(test)
Benchmark System (six parse trees)	77.35	77.92
Benchmark System (top Charniak parse tree only)	74.76	–
NN+SLL+LM2	72.29	74.15
NN+SLL+LM2+Charniak (level 0 only)	74.44	75.65
NN+SLL+LM2+Charniak (levels 0 & 1)	74.50	75.81
NN+SLL+LM2+Charniak (levels 0 to 2)	75.09	76.05
NN+SLL+LM2+Charniak (levels 0 to 3)	75.12	75.89
NN+SLL+LM2+Charniak (levels 0 to 4)	75.42	76.06
NN+SLL+LM2+CHUNK	–	74.72
NN+SLL+LM2+PT0	–	75.49

Increases slowly



SENNA - the final implementation in C

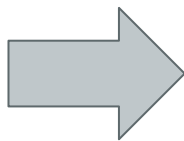
- Uses the best feature-engineered models described above (beat the state of the art), but it's also really fast

POS System	RAM (MB)	Time (s)
Toutanova et al. (2003)	800	64
Shen et al. (2007)	2200	833
SENNA	32	4

SRL System	RAM (MB)	Time (s)
Koomen et al. (2005)	3400	6253
SENNA	124	51

Last Thought

Why ignore these task-specific engineered features? Why abandon it all for neural networks?



No NLP task covers the goals of NLP. Generally, task-specific engineering should not be the end goal of NLP (understanding text completely).

Thanks! Any Questions?
(We know it was a long paper)

Sources

1. Collobert, Ronan, et al. "Natural language processing (almost) from scratch." *Journal of Machine Learning Research* 12.Aug (2011): 2493-2537.
2. Toutanova, Kristina, et al. "Feature-rich part-of-speech tagging with a cyclic dependency network." *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003.
3. Sha, Fei, and Fernando Pereira. "Shallow parsing with conditional random fields." *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics, 2003.
4. Ni, Yepeng, et al. "An indoor pedestrian positioning method using HMM with a fuzzy pattern recognition algorithm in a WLAN fingerprint system." *Sensors* 16.9 (2016): 1447.
5. Lafferty, John, Andrew McCallum, and Fernando CN Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." (2001).