Deep Learning for NLP

Kiran Vodrahalli
Feb 11, 2015

Overview

- What is NLP?
  - Natural Language Processing
  - We try to extract meaning from text: sentiment, word sense, semantic similarity, etc.
- How does Deep Learning relate?
  - NLP typically has sequential learning tasks
- What tasks are popular?
  - Predict next word given context
  - Word similarity, word disambiguation
  - Analogy / Question answering

Papers Timeline

- Bengio (2003)
- Hinton (2009)
- Mikolov (2010, 2013, 2013, 2014)
  - RNN → word vector → phrase vector → paragraph vector
- Quoc Le (2014, 2014, 2014)
- Interesting to see the transition of ideas and approaches (note: Socher 2010 – 2014 papers)
- We will go through the main ideas first and assess specific methods and results in more detail later

Standard NLP Techniques

- Bag-of-Words
- Word-Context Matrices
  - LSA
  - Others... (construct matrix, smooth, dimension reduction)
- Topic modeling
  - Latent Dirichlet Allocation
    - Statistics-based
- N-grams

Some common metrics in NLP

- Perplexity (PPL): Exponential of average negative log likelihood
  - geometric average of the inverse of probability of seeing a word given the previous n words
  - 2 to the power of cross entropy of your language model with the test data
  - $\frac{1}{\hat{P}\left(w_t | w_1^{t-1}\right)}$
- BLEU score: measures how many words overlap in a given translation compared to a reference, with higher scores given to sequential words
  - Values closer to 1 are more similar (would like human and machine translation to be very close)
- Word Error Rate (WER): derived from Levenstein distance
  - WER = (S + D + I)/ (S + D + C)
  - S = substitutions, D = deletions, I = insertions, C = corrections

Statistical Model of Language

- Conditional probability of one word given all the previous ones
-

$$\hat{P}\left(w_1^T\right) = \prod_{t=1}^{T} \hat{P}\left(w_t | w_1^{t-1}\right)$$

Issues for Current Methods

- Too slow
- Stopped improving when fed increasingly larger amounts of data
- Very simple and naïve; works surprisingly well but not well enough
- Various methods don't take into account semantics, word-order, long-range context
- A lot of parsing required and/or hand-built models
- Need to generalize!

N-grams

- Consider combinations of successive words of smaller size and predict see what comes next for all of those.
- Smoothing can be done for new combinations (which do not occur in training set)
- Bengio: we can improve upon this!
  - They don't typically look at contexts > 3 words
  - Words can be similar: n-grams don't use this to generalize when we should be!

Word Vectors

- Concept will show up in a lot of the papers
- The idea is we represent a word by a dense vector in semantic space
- Other vectors close by should be semantically similar
- Several ways of generating them; the papers we will look at generate them with Neural Net procedures

Neural Probabilistic Language Model (Bengio 2003)

- Fight the curse of dimensionality with continuous word vectors and probability distributions
- Feedforward net that both learns word vector representation and a statistical language model simultaneously
- Generalization: “similar” words have similar feature vectors; probability function is smooth function of these values → small change in features induces small change in probability, and we distribute the probability mass evenly to a combinatorial number of similar neighboring sentences every time we see a sentence.

Bengio's Neural Net Architecture

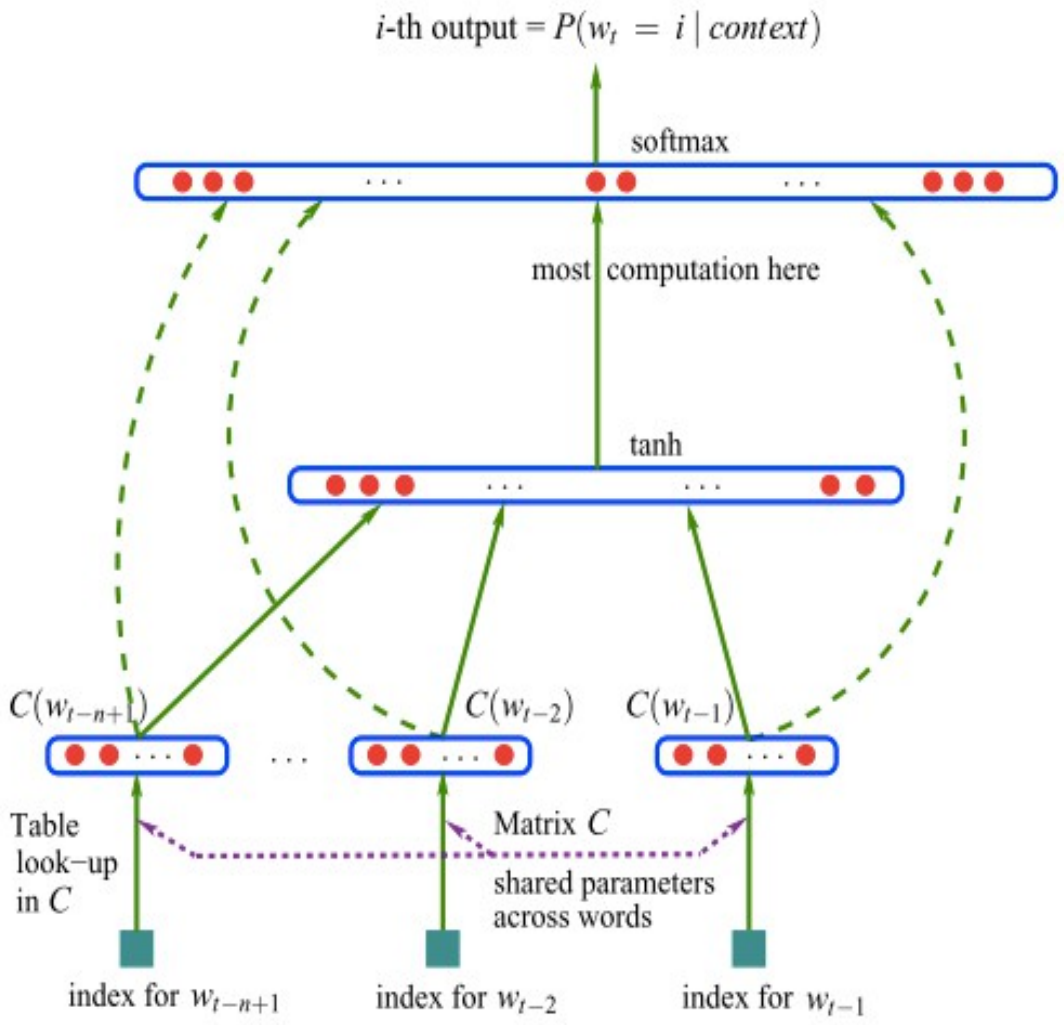


Figure 1: Neural architecture: $f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector.

Bengio Network Performance

- Has lower perplexity than smoothed tri-gram models (weighted sum of probabilities of unigram, bigram, up to trigram) on Brown corpus
- Perplexity of best neural net approach: 252
  - (100 hidden units; look back 4 words; 30 word features, no connections between word layer and output layer; output probability averaged with trigram output probability)
- Perplexity of best tri-gram only approach: 312

RNN-based Language Model (Mikolov 2010)

- RNN-LM: 50% reduction on perplexity possible over n-gram techniques
- Feeding off of Bengio's work, which used feedforward net → Now we try RNN! More general, not as dependent on parsing, morphology, etc. Learn from the data directly.
- Why use RNN?
  - Language data is sequential; RNN is good approach for sequential data (no required fixed input size) → can unrestrict context

Simple RNN Model

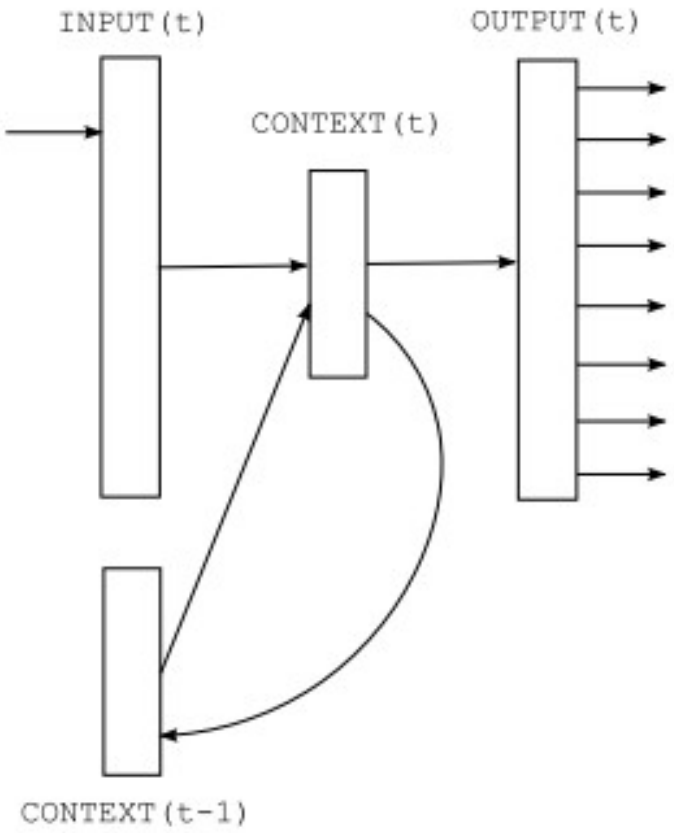


Figure 1: *Simple recurrent neural network.*

RNN Model Description

- Input: x(t): formed by concatenating vector w (current word) with the context s(t – 1)
- Hidden context layer activation: sigmoid
- Output y(t): softmax layer to output probability distribution (we are predicting probability of each word being the next word)
- error(t) = desired(t) – y(t); where desired(t) is 1-of-V encoding for the correct next word
- Word input uses 1-of-V encoding
- Context layer can be initialized with small weights
- Use trunctated backprop through time (BPTT) and SGD to train

More details on RNN model

- Rare word tokens: merge words that occur less often than some threshold into a rare-word token
  - prob(rare word) = $y_{rare}(t)$/ (number of rare words)
  - $y_{rare}(t)$ is the rare-word token
- The dynamic model: network should continue training even during testing phase, since the point of the model is to update the context

Performance of RNN vs. KN5 on WSJ dataset

Table 2: *Comparison of various configurations of RNN LMs and combinations with backoff models while using 6.4M words in training data (WSJ DEV).*

	PPL		WER	
Model	RNN	RNN+KN	RNN	RNN+KN
KN5 - baseline	-	221	-	13.5
RNN 60/20	229	186	13.2	12.6
RNN 90/10	202	173	12.8	12.2
RNN 250/5	173	155	12.3	11.7
RNN 250/2	176	156	12.0	11.9
RNN 400/10	171	152	12.5	12.1
3xRNN static	151	143	11.6	11.3
3xRNN dynamic	128	121	11.3	11.1

More data = larger improvement

Table 1: *Performance of models on WSJ DEV set when increasing size of training data.*

Model	# words	PPL	WER
KN5 LM	200K	336	16.4
KN5 LM + RNN 90/2	200K	271	15.4
KN5 LM	1M	287	15.1
KN5 LM + RNN 90/2	1M	225	14.0
KN5 LM	6.4M	221	13.5
KN5 LM + RNN 250/5	6.4M	156	11.7

More RNN comparisons

- Previous approaches were not state-of-the-art,we display improvement on state-of-the-art AMI system for speech transcription in meetings on NIST RT05 dataset
- Training data: 115 hours of meeting speech from many training corpora
-

Table 4: *Comparison of very large back-off LMs and RNN LMs trained only on limited in-domain data (5.4M words).*

Model	WER static	WER dynamic
RT05 LM	24.5	-
RT09 LM - baseline	24.1	-
KN5 in-domain	25.7	-
RNN 500/10 in-domain	24.2	24.1
RNN 500/10 + RT09 LM	**23.3**	23.2
RNN 800/10 in-domain	24.3	23.8
RNN 800/10 + RT09 LM	23.4	23.1
RNN 1000/5 in-domain	24.2	23.7
RNN 1000/5 + RT09 LM	23.4	22.9
3xRNN + RT09 LM	**23.3**	**22.8**

Mikolov 2013 Summary

- In 2013, word2vec (Google) made big news with word vector representations that were able to represent vector compositionality
- vec(Paris) – vec(France) + vec(Italy) = vec(Rome)
- Trained relatively quickly, NOT using neural net nonlinear complexity
- “less than a day to learn high quality word vectors from 1.6 billion word Google News corpus dataset”
- (note: this corpus internal to Google)

Efficient Estimation of Word Representations in Vector Space (Mikolov 2013)

- Trying to maximize accuracy of vector operations by developing new model architectures that preserve linear regularities among words; minimize complexity
- Approach: continuous word vectors learned using simple model; n-gram NNLM (Bengio) trained on top of these distributed representations
- Extension of previous two papers (Bengio; Mikolov(2010))

Training Complexity

- We are concerned with making the complexity as simple as possible to allow training on larger datasets in smaller amounts of time.
- Definition: O = E*T*Q, where E = # of training epochs, T = # of words in training set, Q = model-specific factor (i.e. in a neural net, counting number of size of connection matrices)
- N: # previous words, D: # dims in representation, H: hidden layer size; V: vocab size
- Feedforward NNLM: $Q = N*D + N*D*H + H*\log_2 V$
- Recurrent NNLM (RNNLM): $Q = H*H + H*\log_2 V$
  - $\log_2 V$ comes from using hierarchical softmax

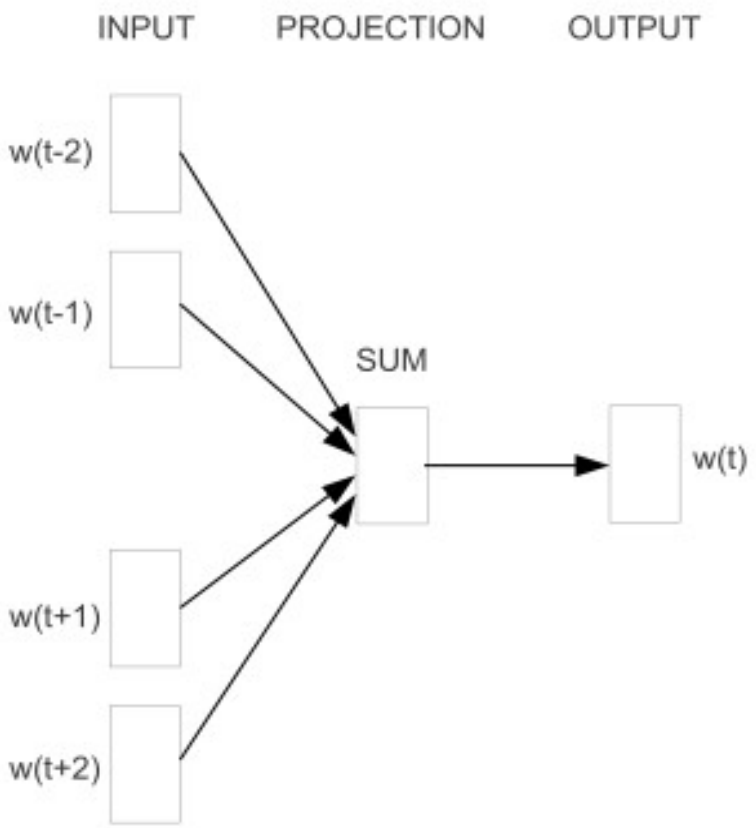
Hierarchical Softmax

- $\frac{e^{z_j}}{\sum_{i=1}^{K} e^{z_k}}$
- Want to learn probability distribution on words
- Speed up calculations by building a conditioning tree
- Tree is Huffman code: high-frequency words are assigned small codes (near the top of the tree)
- Improves updates from V to $\log_2 V$

New Log-linear models

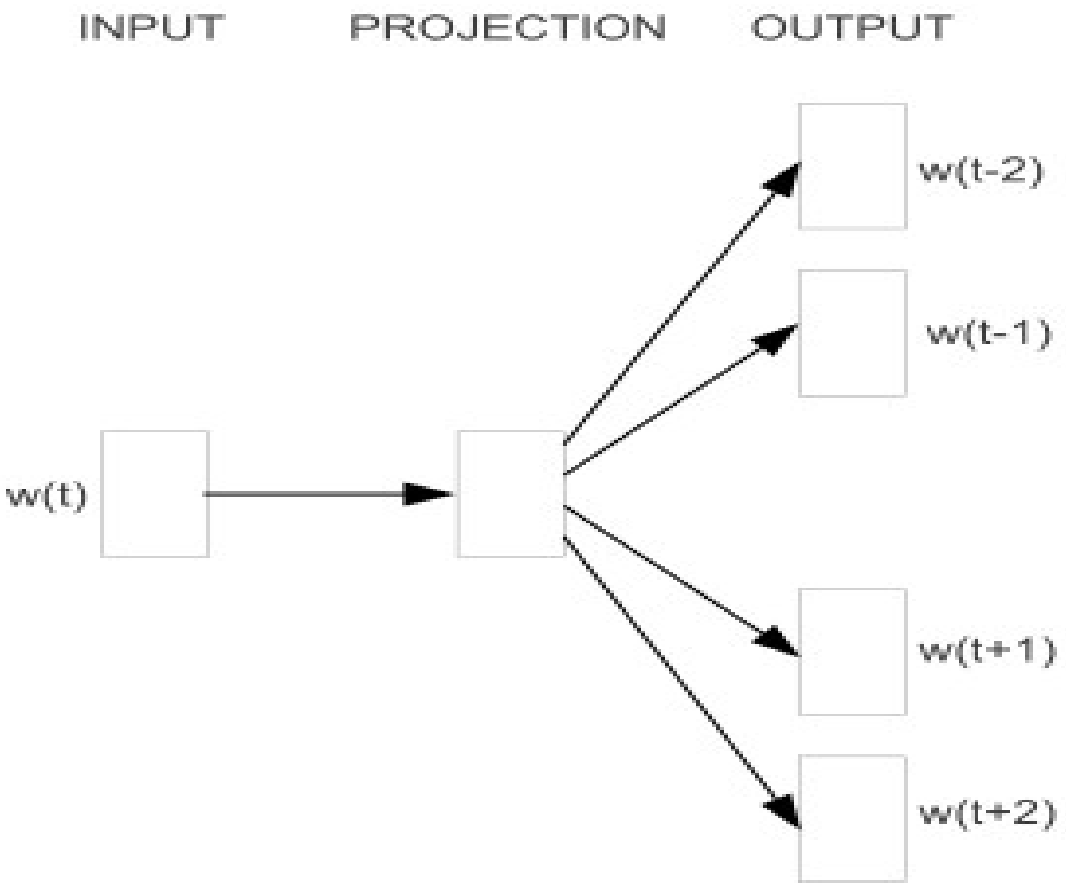
- CBOW (Continuous Bag of Words)
  - Context predicts word
  - All words get projected to same position (averaged) → lose order of words info
  - $Q = N*D + D*\log_2 V$
- Skip-gram (we will go into more detail later)
  - Word predicts context, a range before and after the current word
  - Less weight given to more distant words
  - Log-linear classifier with continuous projection layer
  - C: maximum distance between words
  - $Q = C*(D + D*\log_2 V)$
- avoid the complexity of neural nets to train good word vectors; use log-linear optimization (achieve global maximum on max log probability objective)
- Can take advantage of more data due to speed up

CBOW Diagram



CBOW

Skip-gram diagram



Skip-gram

Results

- Vector algebra result: possible to find answers to analogy questions like “What is the word that is similar to *small* in the same sense as *biggest* is to *big*?” (vec(“biggest”) - vec(“big”) + vec(“small”) = ?)
- The task: test set containing 5 types of semantic questions; 9 types of syntactic questions
- Summarized in the following table:

Mikolov test questions

Table 1: *Examples of five types of semantic and nine types of syntactic questions in the Semantic-Syntactic Word Relationship test set.*

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Performance on Syntactic-Semantic Questions

Table 2: *Accuracy on subset of the Semantic-Syntactic Word Relationship test set, using word vectors from the CBOW architecture with limited vocabulary. Only questions containing words from the most frequent 30k words are used.*

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 3: *Comparison of architectures using models trained on the same data, with 640-dimensional word vectors. The accuracies are reported on our Semantic-Syntactic Word Relationship test set, and on the syntactic relationship test set of [20]*

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Summary comparison of architectures

- Word vectors from RNN perform well on syntactic questions; NNLM vectors perform better than RNN (RNNLM has a non-linear layer directly connected to word vectors; NNLM has interfering projection layer)
- CBOW > NNLM on synactic, bit better on semantic
- Skip-gram ~ CBOW (a bit worse) on syntactic
- Skip-gram >>> everything else on semantic
- This is all for training done with parallel training

Comparison to other approaches (1 CPU only)

Table 4: *Comparison of publicly available word vectors on the Semantic-Syntactic Word Relationship test set, and word vectors from our models. Full vocabularies are used.*

Model	Vector Dimensionality	Training words	Accuracy [%]		
			Semantic	Syntactic	Total
Collobert-Weston NNLM	50	660M	9.3	12.3	11.0
Turian NNLM	50	37M	1.4	2.6	2.1
Turian NNLM	200	37M	1.4	2.2	1.8
Mnih NNLM	50	37M	1.8	9.1	5.8
Mnih NNLM	100	37M	3.3	13.2	8.8
Mikolov RNNLM	80	320M	4.9	18.4	12.7
Mikolov RNNLM	640	320M	8.6	36.5	24.6
Huang NNLM	50	990M	13.3	11.6	12.3
Our NNLM	20	6B	12.9	26.4	20.3
Our NNLM	50	6B	27.9	55.8	43.2
Our NNLM	100	6B	34.2	**64.5**	50.8
CBOW	300	783M	15.5	53.1	36.1
Skip-gram	300	783M	**50.0**	55.9	**53.3**

Varying epochs, training set size

Table 5: *Comparison of models trained for three epochs on the same data and models trained for one epoch. Accuracy is reported on the full Semantic-Syntactic data set.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days]
			Semantic	Syntactic	Total	
3 epoch CBOW	300	783M	15.5	53.1	36.1	1
3 epoch Skip-gram	300	783M	50.0	55.9	53.3	3
1 epoch CBOW	300	783M	13.8	49.9	33.6	0.3
1 epoch CBOW	300	1.6B	16.1	52.6	36.1	0.6
1 epoch CBOW	600	783M	15.4	53.3	36.2	0.7
1 epoch Skip-gram	300	783M	45.6	52.2	49.2	1
1 epoch Skip-gram	300	1.6B	52.2	55.1	53.8	2
1 epoch Skip-gram	600	783M	56.7	54.5	55.5	2.5

Microsoft Sentence Completion

- 1040 sentences; one word missing / sentence, goal is to select the word that is most coherent with the rest of the sentence

Table 7: *Comparison and combination of models on the Microsoft Sentence Completion Challenge.*

Architecture	Accuracy [%]
4-gram [32]	39
Average LSA similarity [32]	49
Log-bilinear model [24]	54.8
RNNLMs [19]	55.4
Skip-gram	48.0
Skip-gram + RNNLMs	**58.9**

Skip-gram Learned Relationships

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Versatility of vectors

- Word vector representation also allows solving tasks like finding the word that doesn't belong in the list (i.e. (“apple”, “orange”, “banana”, “airplane”))
- Compute average vector of words, find the most distant one → this is out of the list.
- Good word vectors could be useful in many NLP applications: sentiment analysis, paraphrase detection

DistBelief Training

- They claim should be possible to train CBOW and Skip-gram models on corpora with ~ 10^12 words, orders of magnitude larger than previous results (log complexity of vocabulary size)
-

Table 6: *Comparison of models trained using the DistBelief distributed framework. Note that training of NNLM with 1000-dimensional vectors would take too long to complete.*

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125

Focusing on Skip-gram

- Skip-gram did much better than everything else on the semantic questions; this is interesting.
- We investigate further improvements (Mikolov 2013, part 2)
- Subsampling gives more speedup
- So does negative sampling (used over hierarchical softmax)

Recall: Skip-gram Objective

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$$

$$p(w_O|w_I)=\frac{\exp\left({v'_{w_O}}^{\top}v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top}v_{w_I}\right)}$$

Basic Skip-gram Formulation

- (Again, we're maximizing average log probability over the set of context words we predict with the current word)
- C is the size of the training context
  - Larger c → more accuracy, more time
- v_w and v_w' are input and output representations of w, W is # of words
- Use softmax function to define probability; this formulation is not efficient → hierarchical softmax

OR: Negative Sampling

- Another approach to learning good vector representations to hierarchical softmax
- Based off of Noise Constrastive Estimation (NCE): a good model should differentiate data from noise via logistic regression
- Simplify NCE → Negative sampling
-

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

Explanation of NEG objective

- For each (word, context) example in the corpus we take k additional samples of (word, context) pairs NOT in the corpus (by generating random pairs according to some distribution Pn(w))
- We want the probability that these are valid to be very low
- These are the “negative samples”; k ~ 5 – 20 for larger data sets, ~ 2 – 5 for small
- $$= \arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} (1 - p(D=1|c,w;\theta))$$

Subsampling frequent words

- Extremely frequent words provide less information value than rarer words
- Each word w_i in training set is discarded with probability; t (threshold) ~ 10^-5: aggressively subsamples while preserving frequency ranking
- Accelerates learning; does well in practice

f is frequency of word; P(w_i): prob to discard

- $$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Results on analogical reasoning (previous paper's task)

- Recall the task: “Germany”: “Berlin” :: “France”:?
- Approach to solve: find x s.t. vec(x) is closest to vec(“Berlin”) - vec(“Germany”) + vec(“France”)
- V = 692K
- Standard sigmoidal RNNs (highly non-linear) improve upon this task; skip-gram is highly linear
- Sigmoidal RNNs → preference for linear structure? Skip-gram may be a shortcut

Performance on task

Method	Time [min]	Syntactic [%]	Semantic [%]	Total accuracy [%]
NEG-5	38	63	54	59
NEG-15	97	63	58	**61**
HS-Huffman	41	53	40	47
NCE-5	38	60	45	53
The following results use 10^{-5} subsampling				
NEG-5	14	61	58	60
NEG-15	36	61	61	**61**
HS-Huffman	21	52	59	55

Table 1: Accuracy of various Skip-gram 300-dimensional models on the analogical reasoning task as defined in [8]. NEG-k stands for Negative Sampling with k negative samples for each positive sample; NCE stands for Noise Contrastive Estimation and HS-Huffman stands for the Hierarchical Softmax with the frequency-based Huffman codes.

What do the vectors look like?

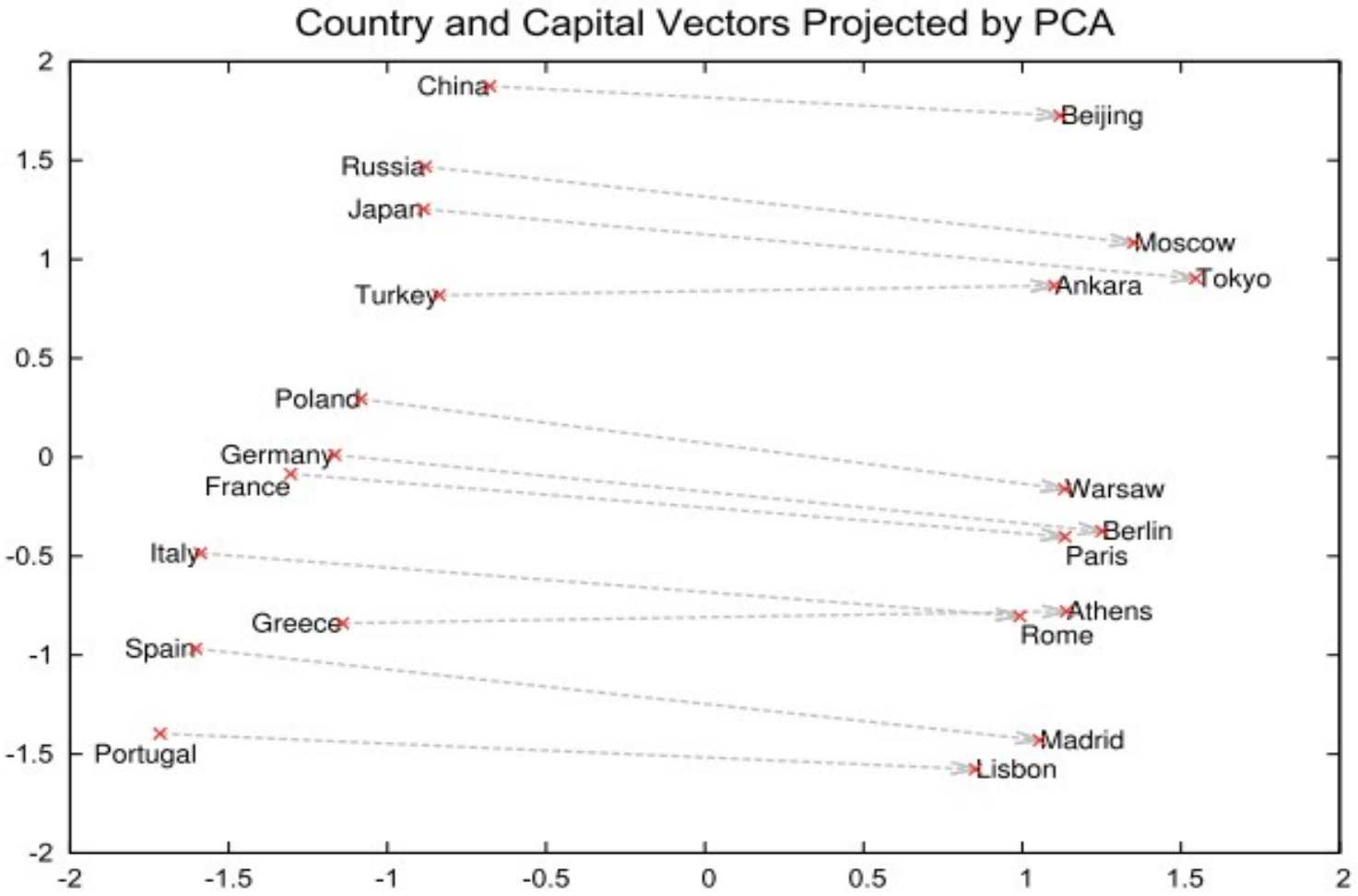


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Applying Approach to Phrase vectors

- "phrase" → meaning can't be found by composition; words that appear frequently together; infrequently elsewhere
- Ex: New York Times becomes a single token
- Generate many "reasonable phrases" using unigram/bigram frequencies with a discount term; (don't just use all n-grams)
- Use Skip-gram for analogical reasoning task for phrases (3128 examples)
-

Method	Dimensionality	No subsampling [%]	10^{-5} subsampling [%]
NEG-5	300	24	27
NEG-15	300	27	42
HS-Huffman	300	19	**47**

Table 3: Accuracies of the Skip-gram models on the phrase analogy dataset. The models were trained on approximately one billion words from the news dataset.

Examples of analogical reasoning task for phrases

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

Table 2: Examples of the analogical reasoning task for phrases (the full test set has 3218 examples). The goal is to compute the fourth phrase using the first three. Our best model achieved an accuracy of 72% on this dataset.

Additive Compositionality

- Can meaningfully combine vectors with term-wise addition
- Examples:

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

Additive Compositionality

- Explanation: word vectors in linear relationship with softmax nonlinearity
- Vectors represent distribution of context in which word appears
- These values are logarithmically related to probabilities, so sums correspond to products; i.e. we are ANDing together the two words in the sum.
- Sum of word vecs ~ product of context distributions

Nearest Neighbors of Infrequent Words

Model (training time)	Redmond	Havel	ninjutsu	graffiti	capitulate
Collobert (50d) (2 months)	conyers lubbock keene	plauen dzerzhinsky osterreich	reiki kohona karate	cheesecake gossip dioramas	abdicate accede rearm
Turian (200d) (few weeks)	McCarthy Alston Cousins	Jewell Arzu Ovitz	- - -	gunfire emotion impunity	- - -
Mnih (100d) (7 days)	Podhurst Harlang Agarwal	Pontiff Pinochet Rodionov	- - -	anaesthetics monkeys Jews	Mavericks planning hesitated
Skip-Phrase (1000d, 1 day)	Redmond Wash. Redmond Washington Microsoft	Vaclav Havel president Vaclav Havel Velvet Revolution	ninja martial arts swordsmanship	spray paint grafitti taggers	capitulation capitulated capitulating

Table 6: Examples of the closest tokens given various well known models and the Skip-gram model trained on phrases using over 30 billion training words. An empty cell means that the word was not in the vocabulary.

Paragraph Vector!

- Quoc Le and Mikolov (2014)
- Input is often required to be fixed-length for NNs
- Bag-of-words lose ordering of words and ignore semantics
- Paragraph Vector is unsupervised algorithm that learns fixed length representation of from variable-length texts: each doc is a dense vector trained to predict words in the doc
- More general than Socher approach (RNTNs)
- New state-of-art: on sentiment analysis task, beat the best by 16% in terms of error rate.
- Text classification: beat bag-of-words models by 30%

The model

- Concatenate paragraph vector with several word vectors (from paragraph) → predict following word in the context
- Paragraph vectors and word vectors trained by SGD and backprop
- Paragraph vector unique to each paragraph
- Word vectors shared over all paragraphs
- **Can construct representations of variable-length input sequences (beyond sentence)**

Paragraph Vector Framework

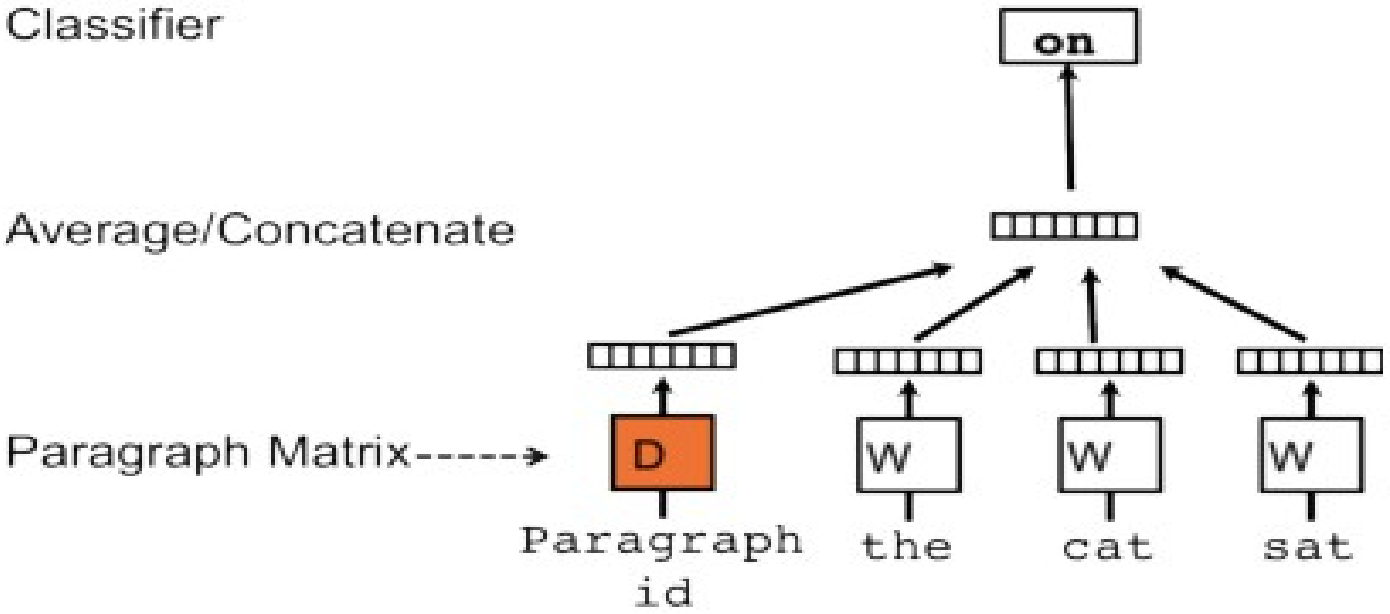


Figure 2. A framework for learning paragraph vector. This framework is similar to the framework presented in Figure 1; the only change is the additional paragraph token that is mapped to a vector via matrix D. In this model, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph.

PV-DM: Distributed Memory Model of Paragraph Vectors

- N paragraphs, M words in vocab
- Each paragraph → p dims; words → q dims
- N*p + M*q; updates during training are sparse
- Contexts are fixed length, sliding window over paragraph; paragraph shared across all contexts which are derived from that paragraph
- Paragraph matrix D; tokens act as memory "what is missing" from current context
- Paragraph vector averaged/concatenated with word vectors to predict next word in context

Model parameters recap

- Word vectors W; softmax weights U, b
- Paragraph vectors D on previously seen paragraphs
- Note: at prediction time, need to calculate paragraph vector for new paragraph. → do gradient descent leaving all other parameters (W, U, b) fixed.
- Resulting vectors can be fed to other ML models

Why are paragraph vectors good

- Learned from unlabeled data
- Take word order into consideration (better than n-gram)
- Not too high-dimensional; generalizes well

Distributed bag of words

- Paragraph vector w/out word order
- Store only softmax weights aside from paragraph vectors
- Force model to predict words randomly sampled from paragraph
- (sample text window, sample word from window and form classification task with vector)
- Analagous to skip-gram model

PV-DBOW picture

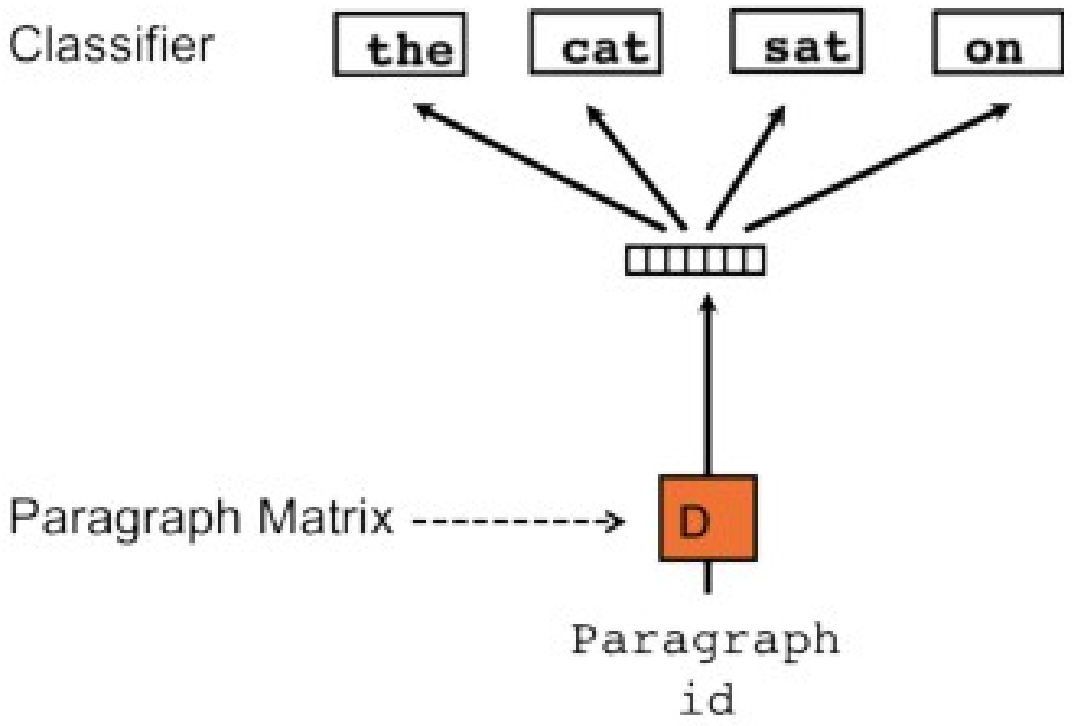


Figure 3. Distributed Bag of Words version of paragraph vectors. In this version, the paragraph vector is trained to predict the words in a small window.

Experiments

- Test with standard PV-DM
- Use combination of PV-DM with PV-DBOW
- Latter typically does better
- Tasks:
  - Sentiment Analysis (Stanford Treebank)
  - Sentiment Analysis (IMDB)
  - Information Retrieval: for search queries, create triple of paragraphs. Two are from query results, one is sampled from rest of collection
    - Which is different?

Experimental Protocols

- Learned vectors have 400 dimensions
- For Stanford Treebank, optimal window size = 8: paragraph vec + 7 word vecs → predict 8th word
- For IMDB, optimal window size = 10
- Cross validate window size between 5 and 12
- Special characters treated as normal words

Stanford Treebank Results

Table 1. The performance of our method compared to other approaches on the Stanford Sentiment Treebank dataset. The error rates of other methods are reported in (Socher et al., 2013b).

Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	**12.2%**	**51.3%**

IMDB Results

Table 2. The performance of Paragraph Vector compared to other approaches on the IMDB dataset. The error rates of other methods are reported in (Wang & Manning, 2012).

Model	Error rate
BoW (bnc) (Maas et al., 2011)	12.20 %
BoW (bΔt'c) (Maas et al., 2011)	11.77%
LDA (Maas et al., 2011)	32.58%
Full+BoW (Maas et al., 2011)	11.67%
Full+Unlabeled+BoW (Maas et al., 2011)	11.11%
WRRBM (Dahl et al., 2012)	12.58%
WRRBM + BoW (bnc) (Dahl et al., 2012)	10.77%
MNB-uni (Wang & Manning, 2012)	16.45%
MNB-bi (Wang & Manning, 2012)	13.41%
SVM-uni (Wang & Manning, 2012)	13.05%
SVM-bi (Wang & Manning, 2012)	10.84%
NBSVM-uni (Wang & Manning, 2012)	11.71%
NBSVM-bi (Wang & Manning, 2012)	8.78%
Paragraph Vector	**7.42%**

Information Retrieval Results

Table 3. The performance of Paragraph Vector and bag-of-words models on the information retrieval task. "Weighted Bag-of-bigrams" is the method where we learn a linear matrix W on TF-IDF bigram features that maximizes the distance between the first and the third paragraph and minimizes the distance between the first and the second paragraph.

Model	Error rate
Vector Averaging	10.25%
Bag-of-words	8.10 %
Bag-of-bigrams	7.28 %
Weighted Bag-of-bigrams	5.67%
Paragraph Vector	**3.82%**

Takeaways of Paragraph Vector

- PV-DM > PV-DBOW; combination is best
- Concatenation > sum in PV-DM
- Paragraph vector computation can be expensive, but is do-able. For testing, the IMDB dataset (25,000 docs, 230 words/doc)
- For IMDB testing, paragraph vectors were computed in parallel 30 min using 16 core machine
- This method can be applied to other sequential data too

Neural Nets for Machine Translation

- Machine translation problem: you have a source sentence in language A and a target language B to derive
- Translate A → B: hard, large # of possible translations
- Typically there is a pipeline of techniques
- Neural nets have been considered as component of pipeline
- Lately, go for broke: why not do it all with NN?
- Potential weakness: fixed, small vocab

Sequence-to-Sequence Learning (Sutskever, Vinyals, Le 2014)

- Main problem with deep neural nets: can only be applied to problems with inputs and targets of fixed dimensionality
- RNNs do not have that constraint, but have fuzzy memory
- LSTM is a model that is able to keep long-term context
- LSTMs are applied to English to French translation (sequence of english words → sequence of french words)

How are LSTMs Built?

(references to Graves (2014))

Basic RNN: “Deep learning in time and space”

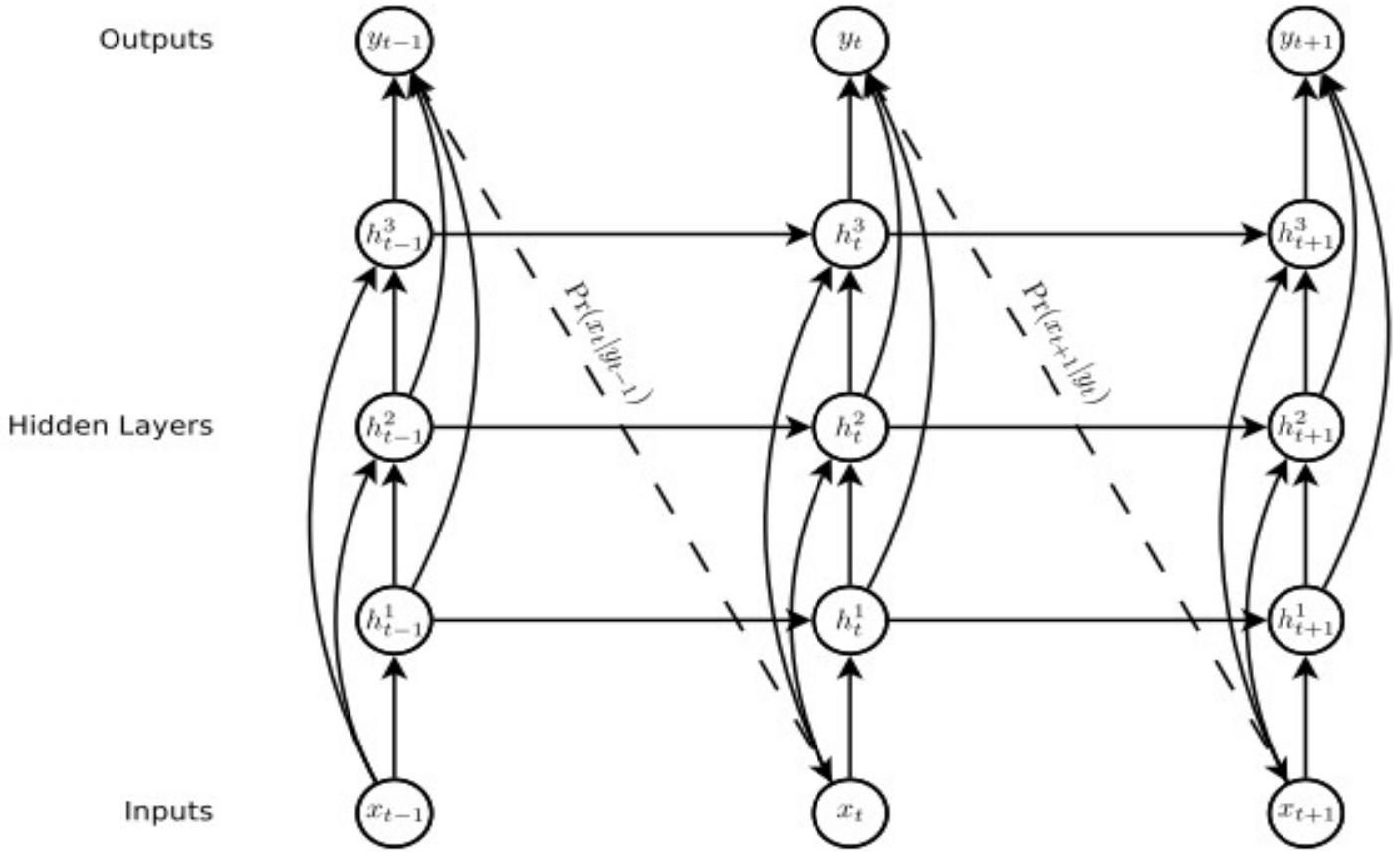


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.

LSTM Memory Cells

- Instead of hidden layer being element-wise application of sigmoid function, we custom design “memory cells” to store information
- These end up being better at finding / exploiting long-range dependencies in data

LSTM block

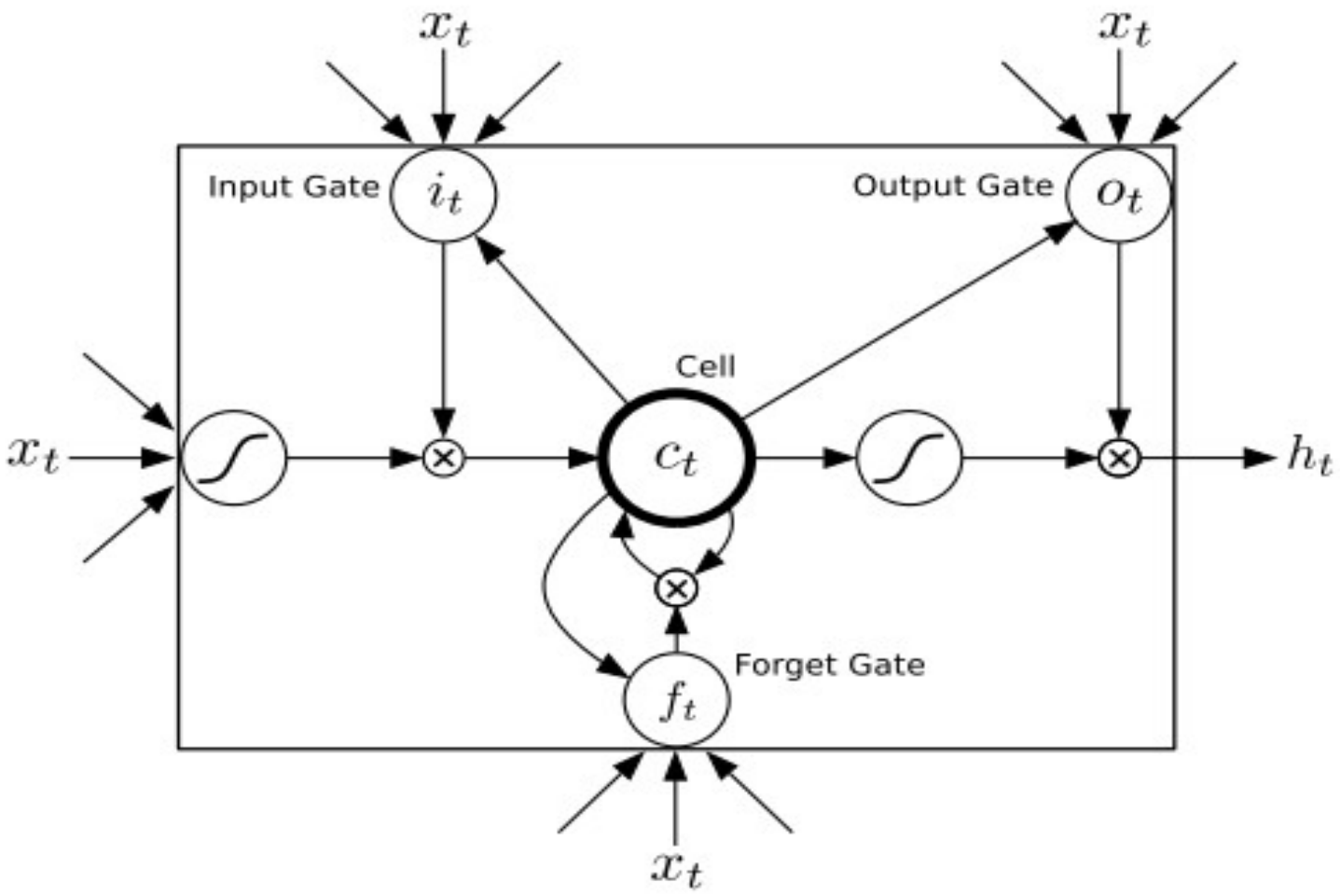


Figure 2: **Long Short-term Memory Cell**

LSTM equations

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right) \quad (7)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right) \quad (8)$$

$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right) \quad (9)$$

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right) \quad (10)$$

$$h_t = o_t \tanh(c_t) \quad (11)$$

i_t: input gate, f_t: forget gate, c_t: cell, o_t: output gat, h_t: hidden vector

Model in more detail

- Deep LSTM1 maps input sequence to large fixed-dimension vector; reads input 1 time step at a time
- Deep LSTM2: decodes target sequence from fixed-dimension vector (essentially RNN-LM conditioned on input sequence)
- Goal of LSTM: estimate conditional probability p(yT' | xT), where xT is the sequence of english words (length T) and yT' is a translation to french (length T'). Note T != T' necessarily.

LSTM translation overview

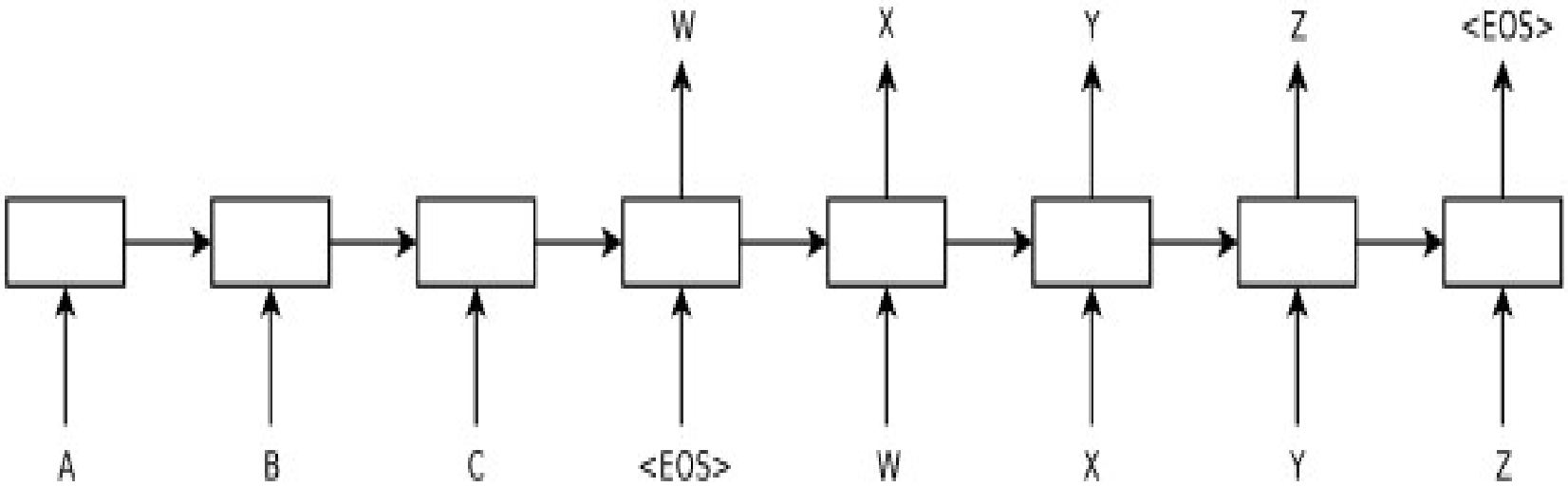


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

Model continued (2)

- Probability distributions represented with softmax
- . v is fixed dimensional representation of input xT

$$p(y_1, \ldots, y_{T'} | x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \ldots, y_{t-1})$$

Model continued (3)

- Different LSTMs were used for input and output (trained with different resulting weights) → can train multiple language pairs as a result
- LSTMs had 4 layers
- In training, reversed the order of the input phrase (the english phrase).
- If <a, b, c> corresponds to <x, y, z>, then the input was fed to LSTM as: <c, b, a> → <x, y, z>
- This greatly improves performance

Experiment Details

- WMT '14 English-French dataset: 348M French Words, 304M English words
- Fixed vocabulary for both languages:
  - 160000 english words, 80000 french words
  - Out of vocab: replaced with <unk>
- Objective: maximize log probability of correct translation T given source sentence S
- Produce translations by finding the most likely one according to LSTM using beam-search decoder (B partial hypotheses at any given time)

Training Details

- Deep LSTMs with 4 layers; 1000 cells/layer; 1000-dim word embeddings
- Use 8000 real #s to represent sentence
  - (4*1000) *2
- Use naïve softmax for output
- 384M parameters; 64M are pure recurrent connections (32M for encoder and 32M for decoder)

Experiment 2

- Second task: Took an SMT system's 1000-best outputs and re-ranked them with the LSTM
- Compute log probability of each hypothesis and average previous score with LSTM score; re-order.

More training details

- Parameter init uniform between -0.08 and 0.08
- Stochastic gradient descent w/out momentum (fixed learning rate of 0.7)
- Halved learning rate each half-epoch after 5 training epochs; 7.5 total epochs for training
- 128-sized batches for gradient descent
- Hard constraint on norm of gradient to prevent explosion
- Ensemble: random initializations + random mini-batch order differentiate the nets

BLEU score: reminder

- Between 0 and 1 (or 0 and 100 → multiply by 100)
- Closer to 1 means better translation
- Basic idea: given candidate translation, get the counts for each of the 4-grams in the translation
- Find max # of times each 4-gram appears in any of the reference translations, and calculate the fraction for 4-gram x: (#x in candidate translation)/(max#x in any reference translation)
- Take geometric mean to obtain total score

Results (BLEU score)

Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

Table 1: The performance of the LSTM on WMT'14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.

Method	test BLEU score (ntst14)
Baseline System [29]	33.30
Cho et al. [5]	34.54
Best WMT'14 result [9]	**37.0**
Rescoring the baseline 1000-best with a single forward LSTM	35.61
Rescoring the baseline 1000-best with a single reversed LSTM	35.85
Rescoring the baseline 1000-best with an ensemble of 5 reversed LSTMs	**36.5**
Oracle Rescoring of the Baseline 1000-best lists	~45

Table 2: Methods that use neural networks together with an SMT system on the WMT'14 English to French test set (ntst14).

Results (PCA projection)

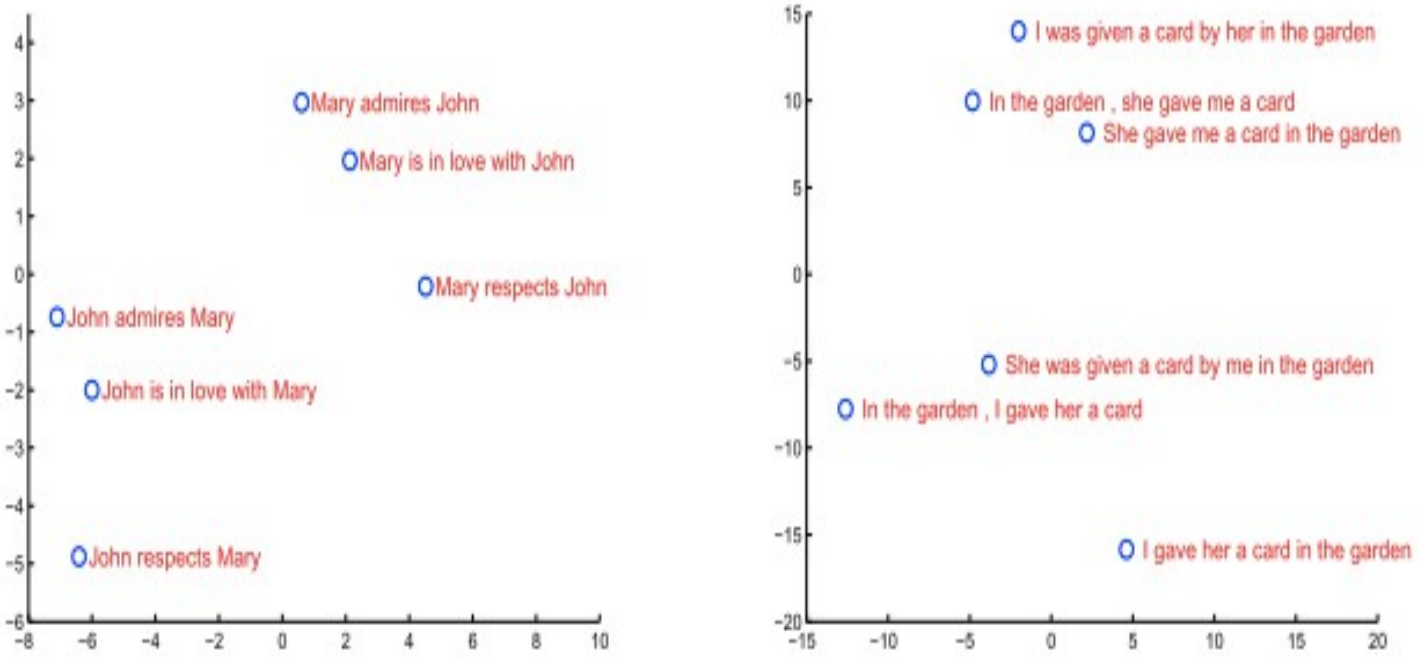


Figure 2: The figure shows a 2-dimensional PCA projection of the LSTM hidden states that are obtained after processing the phrases in the figures. The phrases are clustered by meaning, which in these examples is primarily a function of word order, which would be difficult to capture with a bag-of-words model. Notice that both clusters have similar internal structure.

Performance v. length; rarity

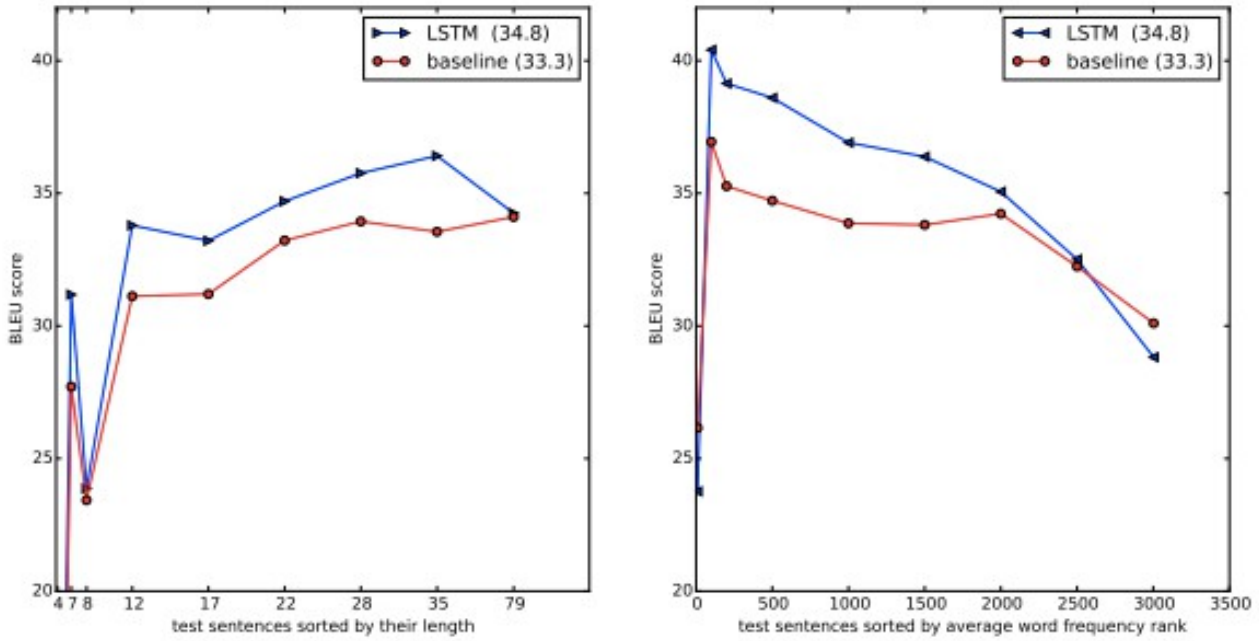


Figure 3: The left plot shows the performance of our system as a function of sentence length, where the x-axis corresponds to the test sentences sorted by their length and is marked by the actual sequence lengths. There is no degradation on sentences with less than 35 words, there is only a minor degradation on the longest sentences. The right plot shows the LSTM's performance on sentences with progressively more rare words, where the x-axis corresponds to the test sentences sorted by their "average word frequency rank".

Results Summary

- LSTM did well on long sentences
- Did not beat the very best WMT'14 system, first time that pure neural translation outperforms an SMT baseline on a large-scale task by a wide margin, even though the LSTM model does not handle out-of-vocab terms
- Improvement by reversing the word order
  - Couldn't train RNN model on non-reversed problem
  - Perhaps is possible with reversed model
- Short-term dependencies important for learning

Rare Word Problem

- In the Neural Machine Translation system we just saw, we had a small vocabulary (only 80k)
- How to handle out-of-vocab (OOV) words?
- Same authors + a few others from previous paper decided to upgrade their previous paper with a simple word alignment technique
- Matches OOV words in target to corresponding word in source, and does a lookup using dictionary

Rare Word Problem (2)

- Previous paper observes sentences with many rare words are translated much more poorly than sentences containing mainly frequent words
- (contrast with Paragraph vector, where less frequent vectors added more information → recall paragraph vector was unsupervised)
- Potential reason prev paper didn't beat standard MT systems: did not take advantage of larger vocabulary and explicit alignments/ phrase counts → fail on rare words

How to solve rare word for NMT?

- Previous paper: use <unk> symbol to represent all OOV words

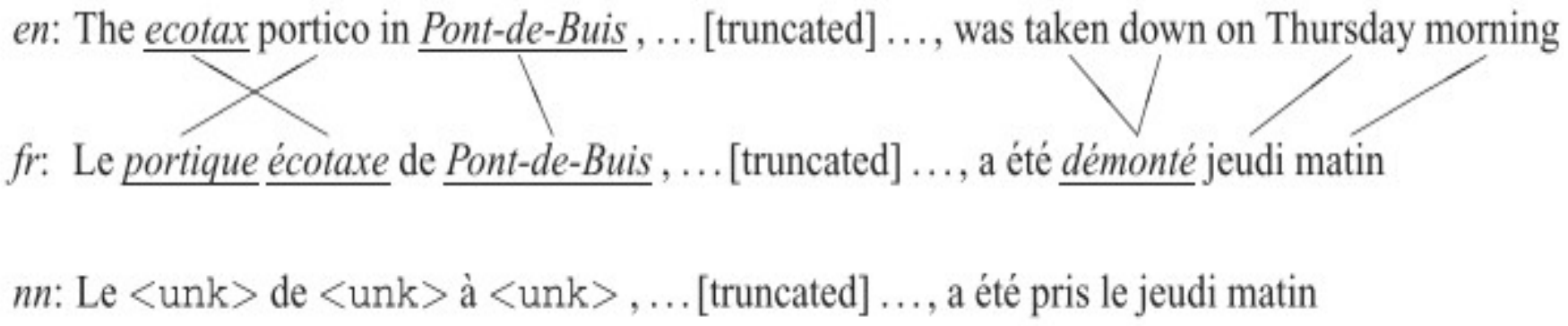


Figure 1: **Example of the rare word problem** – An English source sentence (*en*), a human translation to French (*fr*), and a translation produced by one of our neural network systems (*nn*) before handling OOV words. We highlight *words* that are unknown to our model. The token <unk> indicates an OOV word. We also show a few important alignments between the pair of sentences.

How to solve – intelligently!

- Main idea: match the <unk> outputs with the word that caused them in the source sentence
- Now we can do a dictionary lookup and translate the source word
- If that fails, we can use identity map → just stick the word in from source language (might be the same in both languages → typically for something like a proper noun)

Construct Dictionary

- First we need to align the parallel texts
  - Do this with an unsupervised aligner (Berkeley aligner, GIZA++ tools exist..)
  - General idea: can use expectation maximization on parallel corpora
  - Learn statistical models of the language, find similar features in the corpora and align them
  - A field unto itself
- We DO NOT use the neural net to do any aligning!

Constructing Dictionary (2)

- Three strategies for annotating the texts
- we're modifying the text based on alignment understanding
- They are:
  - Copyable Model
  - PosAll Model (Positional All)
  - PosUnk Model (Positional Unknown)

Copyable Model

- Order unknown words unk1,... in source
- For unknown – unknown matches, use unk1, 2, etc.
- For unknown – known matches, use unk_null (cannot translate unk_null)
- Also use null when no alignment

en: The unk_1 portico in unk_2 …

fr: Le unk_n unk_1 de unk_2 …

Figure 2: **Copyable Model** – an annotated example under the copyable model with two types of unknown tokens: (a) "copyable" tokens, e.g., unk_1, unk_2, etc., and (b) null token unk_n.

PosAll Model

- Only use <unk> token
- In target sentence, place a pos_d token before every <unk>
- pos_d denotes relative position that the target word is aligned to in source (|d| <= 7)

en: The <unk> portico in <unk> …

fr: Le $\underline{pos}_0$ <unk> $\underline{pos}_{-1}$ <unk> $\underline{pos}_1$ de $\underline{pos}_n$ <unk> $\underline{pos}_{-1}$ …

Figure 3: **Positional All Model** – the annotation of the PosAll model, where each word is followed by the relative positional tokens pos_d or the null token $\underline{pos}_n$.

PosUnk Model

- Previous model doubles length of target sentence..
- Let's only annotate alignments of unknown words in target
- Use unkpos_d (|d| <= 7): denote unknown and relative distance to aligned source word (d set to null when no alignment)
- Use <unk> for all other source unknowns

PosUnk Model

en: The <unk> portico in <unk> …

fr: Le $\underline{unkpos_1}$ $\underline{unkpos_{-1}}$ de $\underline{unkpos_1}$ …

Figure 4: **Positional Unknown Model** – the annotations under the PosUnk model, where we annotate only the aligned unknown words with the $unkpos_d$ tokens.

Training

- Train on same dataset as previous paper for comparison with same NN model (LSTM)
- They have difficult with softmax slowness on vocabulary, so they limit to 40K most used french words (reduced from 80k) (only on the output end)
- (they could have used hierarchical softmax or Negative sampling)
- On source side, they use 200K most frequent words
- ALL OTHER WORDS ARE UNKNOWN
- They used the previously-mentioned Berkeley aligner in default

Results

System	Vocab	Corpus	BLEU
Existing state of the art [7]	All	36M	**37.0**
Standard MT + neural components			
LIUM [19] – neural language model	All	12M	33.3
Cho et al. [5] – phrase table neural features	All	12M	34.5
Sutskever et al. [22] – ensemble 5 LSTMs, reranking	All	12M	36.5
Existing end-to-end NMT systems			
Bahdanau et al. [2] – bi-directional gated single RNN	30K	12M	28.5
Sutskever et al. [22] – single LSTM	80K	12M	30.6
Sutskever et al. [22] – ensemble of 5 LSTMs	80K	12M	34.8
Our end-to-end NMT systems			
Single LSTM with 4 layers	40K	12M	29.5
Single LSTM with 4 layers + PosUnk	40K	12M	31.8 (+2.3)
Single LSTM with 6 layers	40K	12M	30.4
Single LSTM with 6 layers + PosUnk	40K	12M	32.7 (+2.3)
Ensemble of 8 LSTMs	40K	12M	34.1
Ensemble of 8 LSTMs + PosUnk	40K	12M	36.9 (+2.8)
Single LSTM with 6 layers	80K	36M	31.5
Single LSTM with 6 layers + PosUnk	80K	36M	33.1 (+1.6)
Ensemble of 8 LSTMs	80K	36M	35.6
Ensemble of 8 LSTMs + PosUnk	80K	36M	**37.5 (+1.9)**

Table 1: **Translation results on newstest2014** – BLEU scores of various systems which differ in terms of: (a) the architecture, (b) the size of the vocabulary used, and (c) the training corpus, either using the full WMT'14 corpus of 36M sentence pairs or a subset of it with 12M pairs. We highlight the performance of our best system in bolded text and state the improvements obtained by our technique of handling rare words (namely, with the PosUnk model). Notice that the more accurate systems achieve a greater improvement from the post-processing step. This is the case because the larger, more accurate models are also more accurate in their output of the alignment information of the unknown word, which makes the post-processing more useful.

Results (2)

- Interesting to note that ensemble models get more gain from the post-processing step
- More larger models identify source word position more accurately → PosUnk more useful
- Best result outperforms currently existing state-of-the-art
- Way outperforms previous NMT systems

And now for something completely different..

- Semantic Hashing – Salakhutdinov & Hinton (2007)
- Finding binary codes for fast document retrieval
- Learn a deep generative model:
  - Lowest layer is word-count vector
  - Highest is a learned binary code for document
- Use autoencoders

TF-IDF

- Term frequency-inverse document frequency
- Measures similarity between documents by comparing word-count vectors
- ~ freq(word in query)
- ~ log(1/freq(word in docs))
- Used to retrieve documents similar to a query document

Drawbacks of TF-IDF

- Can be slow for large vocabularies
- Assumes counts of different words are independent evidence of similarity
- Does not use semantic similarity between words
- Other things tried: LSA, pLSA, → LDA
- We can view as follows: hidden topic variables have directed connections to word-count variables

Semantic hashing

- Produces shortlist of documents in time independent of the size of the document collection; linear in size of shortlist
- The main idea is that learned binary projections are a powerful way to index large collections according to content
- Formulate projections to ~ preserve a similarity function of interest
- Then can explore Hamming ball volume around a query, or use hash tables to search data
- (radius d: differs in at most d positions)

Semantic Hashing (cont.)

- Why binary? By carefully choosing information for each bit, can do better than real-values
- Outline of approach:
  - Generative model for word-count vectors
  - Train RBMs recursively based on generative model
  - Fine-tune representation with multi-layer autoencoder
  - Binarize output of autoencoder with deterministic Gaussian noise

The Approach

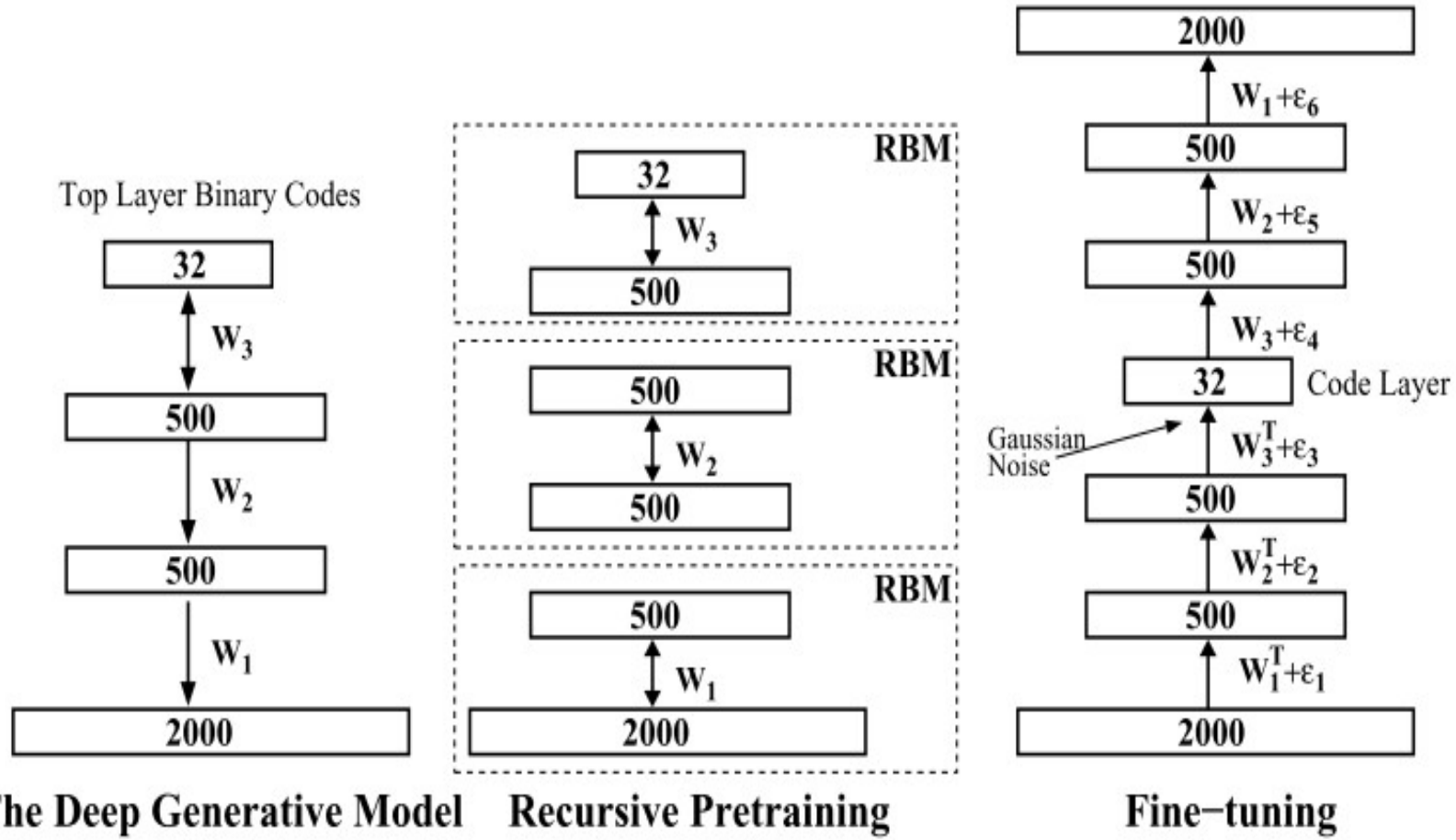
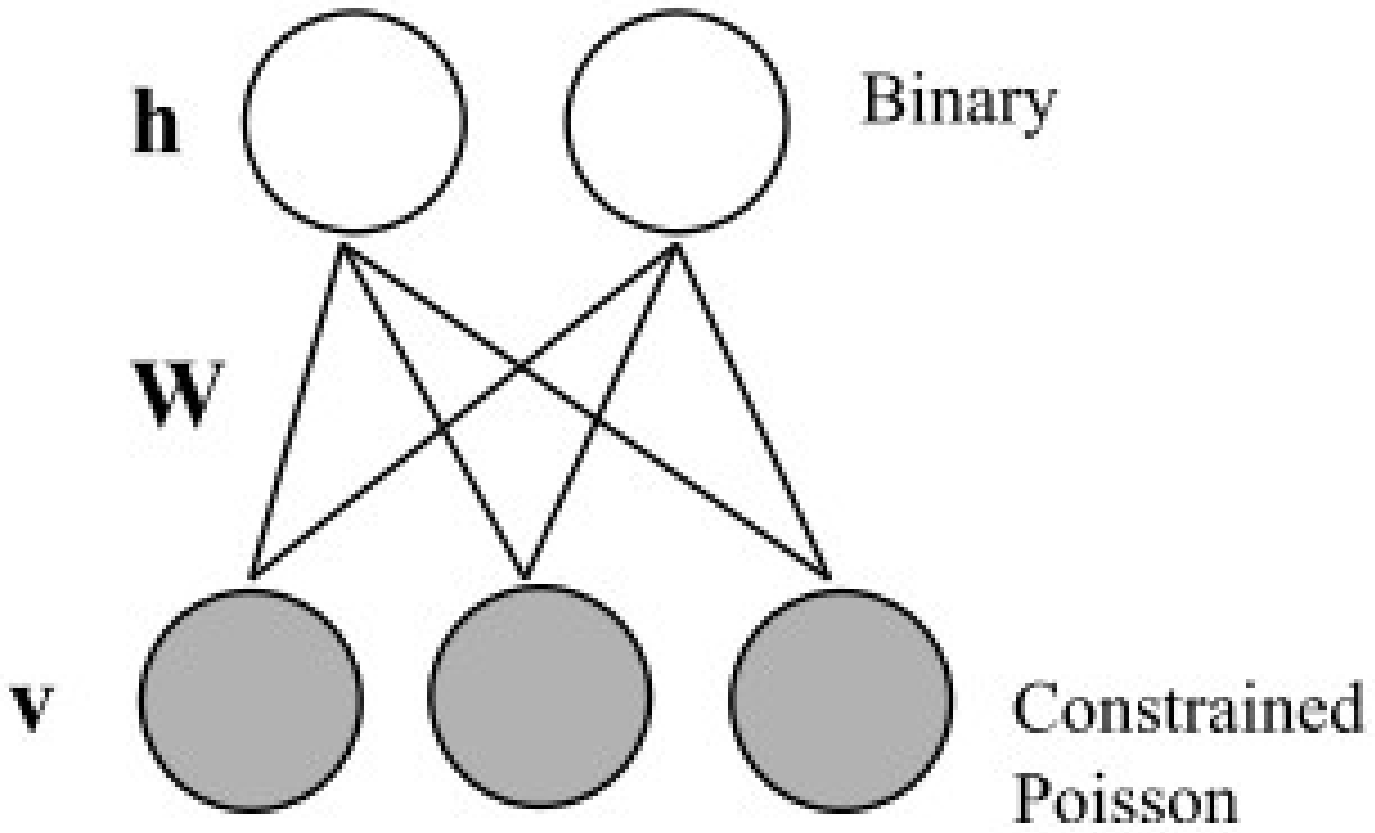


Figure 2: Left panel: The deep generative model. Middle panel: Pretraining consists of learning a stack of RBM's in which the feature activations of one RBM are treated as data by the next RBM. Right panel: After pretraining, the RBM's are "unrolled" to create a multi-layer autoencoder that is fine-tuned by backpropagation.

Modeling word-count vectors

- Constrained Poisson for modeling word count vectors v
  - Ensure mean Poisson rates across all words sum to length of document
  - Learning is stable; deals appropriately w/diff length documents
- Conditional Bernoulli for modeling hidden topic features

First Layer: Poisson → Binary



Model equations

$$p(v_i = n|\mathbf{h}) = \mathrm{Ps}\left(n, \frac{\exp\left(\lambda_i + \sum_j h_j w_{ij}\right)}{\sum_k \exp\left(\lambda_k + \sum_j h_j w_{kj}\right)} N\right) \quad (1)$$

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_i w_{ij} v_i\right) \quad (2)$$

$$\mathrm{Ps}(n, \lambda) = e^{-\lambda}\lambda^n/n!, \sigma(x) = 1/(1+e^{-x})$$

Marginal distribution p(v) w/ energy

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \frac{\exp\left(-E(\mathbf{v}, \mathbf{h})\right)}{\sum_{\mathbf{u}, \mathbf{g}} \exp\left(-E(\mathbf{u}, \mathbf{g})\right)}$$

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i \lambda_i v_i + \sum_i \log\left(v_i!\right) - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

Gradient Ascent Updates/approximation

$$\Delta w_{ij} = \epsilon \frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \epsilon(<v_i h_j>_{data} - <v_i h_j>_{model})$$

$$\Delta w_{ij} = \epsilon(<v_i h_j>_{data} - <v_i h_j>_{recon})$$

Pre-training: Extend beyond one layer

- Now we have the first layer, from Poisson word-count vector to first binary layer.
- Note that this defines an undirected model p(v, h)
- The next layers will all be binary → binary
- p(v) (higher level RBM) starts out as p(h) from lower level, train using data generated from p(h|v) applied to the training data..
- By some variational bound math, this consistently increases lower bound on log probability (which is good)

Summary so far

- Pre-training: We're using higher-level RBMs to improve our deep hierarchical model
- Higher level RBMs are binary → binary
- First level is Poisson → binary
- The point of all this is to initialize weights in the autoencoder to learn a 32-dim representation
- The idea is that this pretraining finds a good area of parameter space (based on the idea that we have a nice generative model)

The Autoencoder

- Autoencoder teaches an algorithm to learn an identity function with reduced dimensionality
- Think of it as forcing the neural net to encapsulate as much information as possible in the smaller # of dimensions so that it can reconstruct it as best as it can
- We use backpropagation here to train word-count vectors with previous architecture (error data comes from itself); divide by N to get probability distribution
- Use cross-entropy error with softmax output

Binarizing the code

- We want the codes found by the autoencoder to be as close to binary as possible
- Add noise: best way to communicate info in presence of noise is to boost your signals so that they are distinguishable → i.e. one strong positive, one strong negative signal → binary
- Don't want noise to mess up training, so we keep it fixed → “deterministic noise”
- Use N(0, 16)

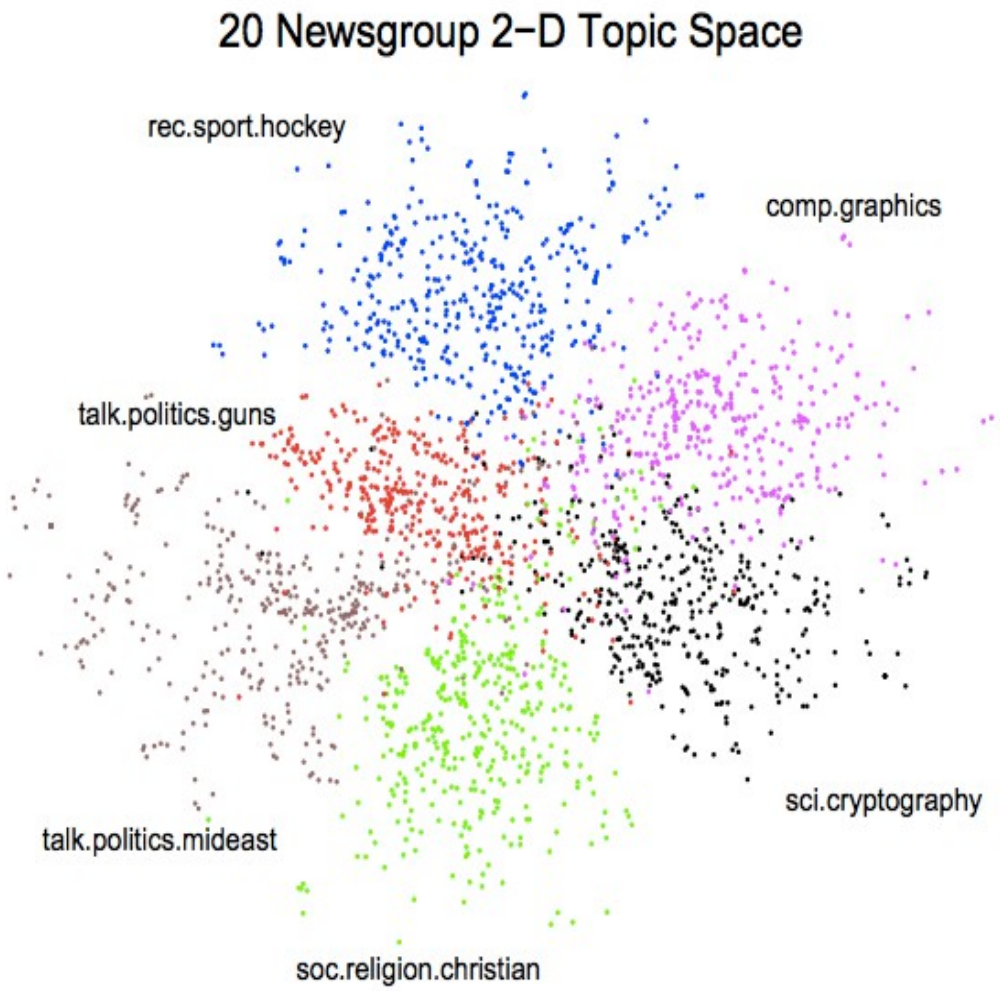
Testing

- The task: given a query document, retrieve relevant documents
- Recall = # retrieved relevant docs/ total relevant docs
- Precision = # relevant retrieved docs / total retrieved docs
- Relevance = check if the documents have the same class label
- LSA and TF-IDF are used as benchmarks

Corpora

- 20-Newsgroups
  - 18845 postings from Usenet
  - 20 different topics
  - Only considered 2000 most frequent words in training
- Reuters Corpus Vol II
  - 804414 newswire stories, 103 topics
  - Corporate/industrial, econ, gov/soc, markets
  - Only considered 2000 most frequent words in training

Results (128-bit)



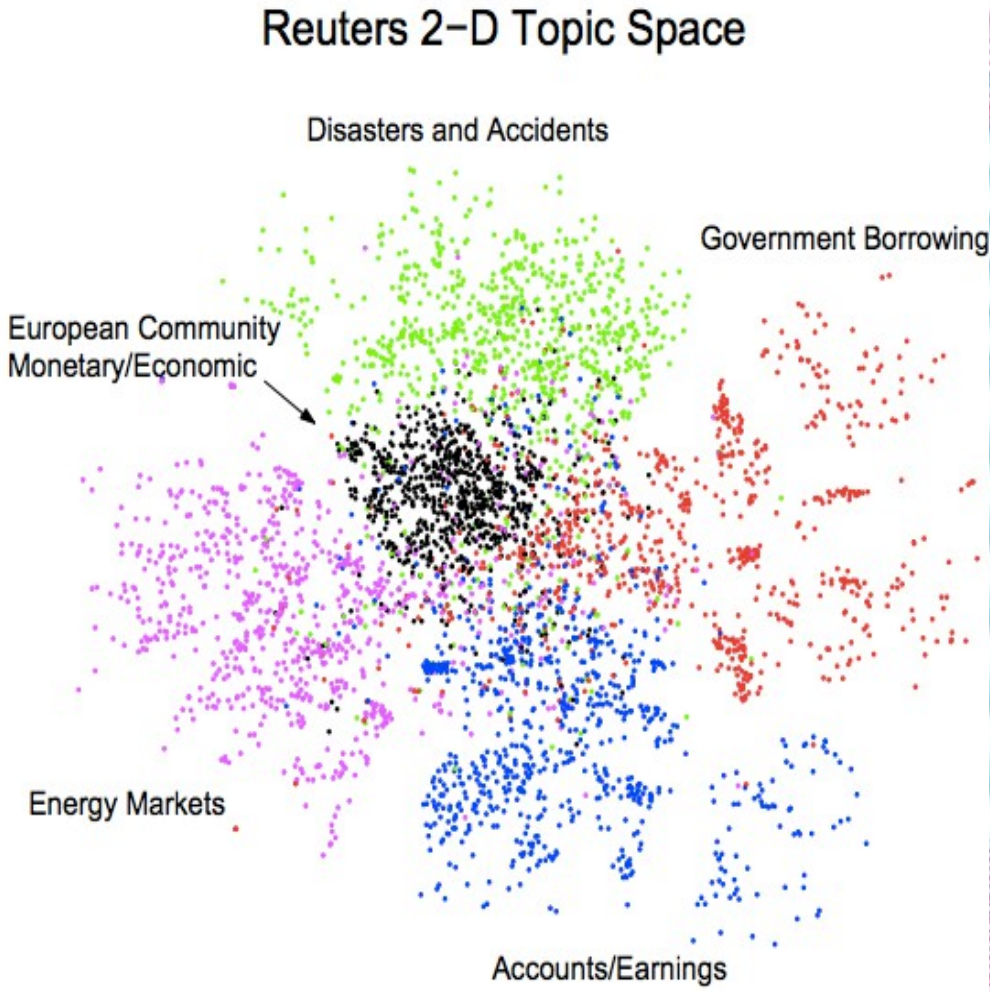


Figure 5: A 2-dimensional embedding of the 128-bit codes using stochastic neighbor embedding for the 20 Newsgroups data (left panel) and t Reuters RCV2 corpus (right panel). See in color for better visualization.

Precision-Recall Curves

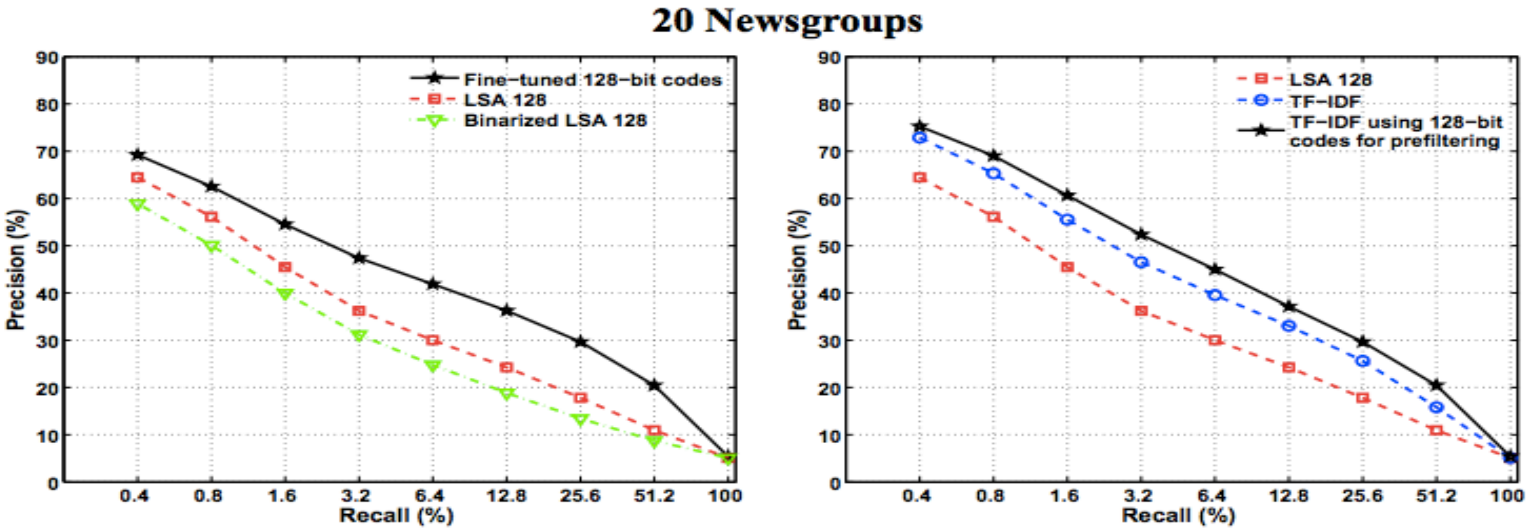


Figure 6: Precision-Recall curves for the 20 Newsgroups dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 7,531 possible queries.

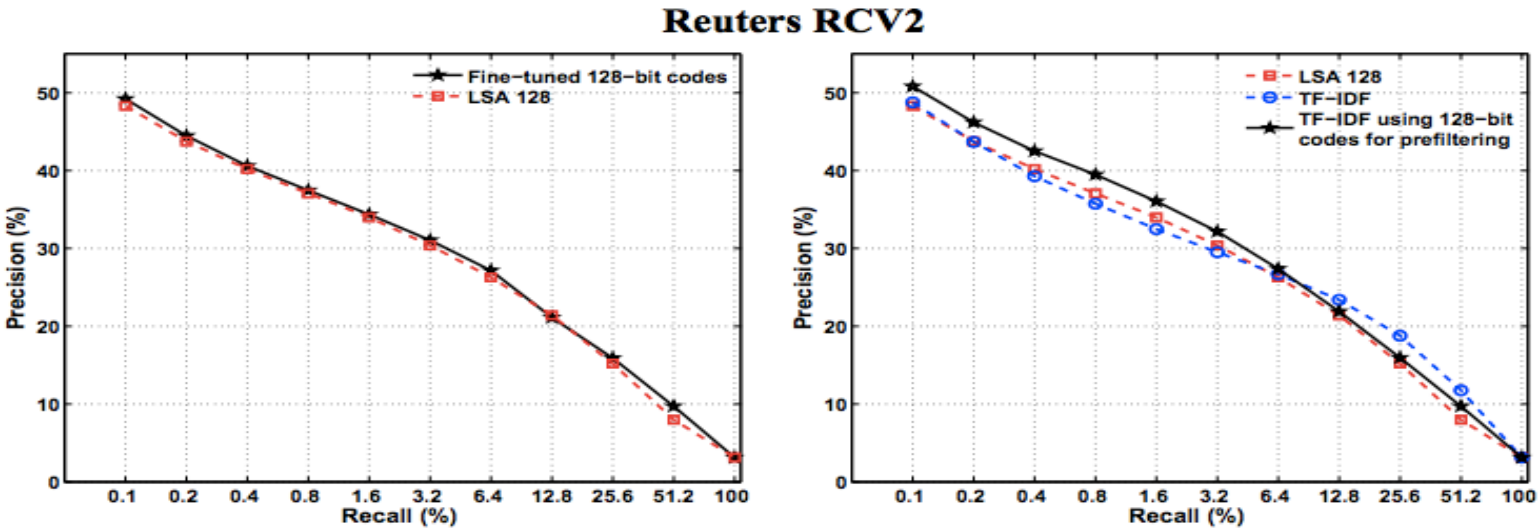
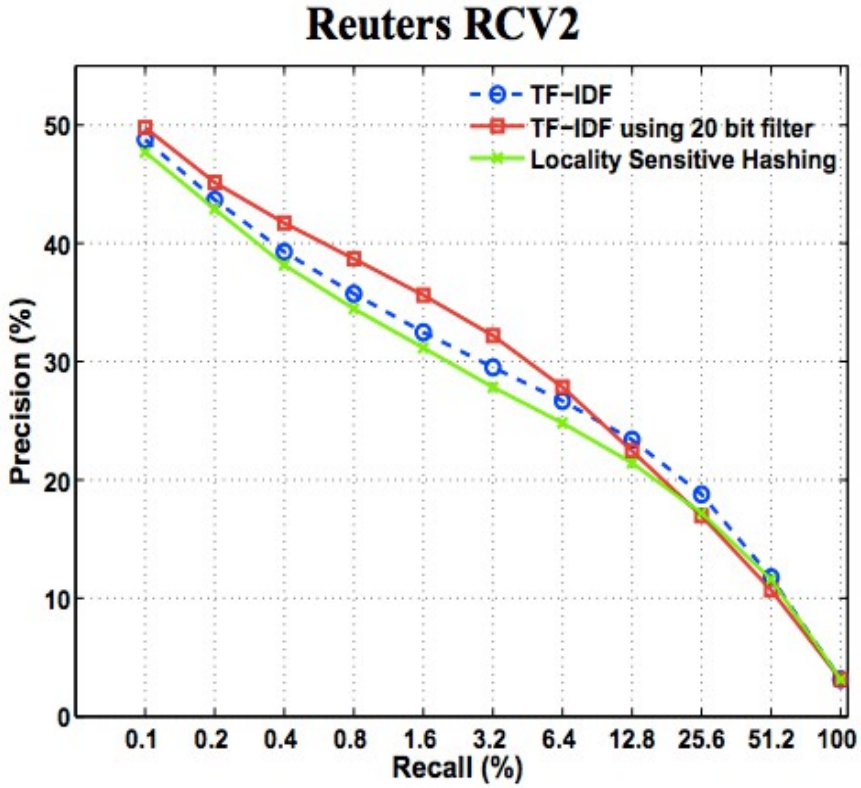


Figure 7: Precision-Recall curves for the Reuters RCV2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries.

Results (20-bit)

- Restricting the bit size down to only 20 bits, does it still work well? (0.4 docs / address)
- Given: query → compute 20bit address
  - > retrieve all documents in Hamming Ball of radius 4 (~ 2500 documents)
  - > No search performed
  - > short list made with TF-IDF
  - > no precision or recall lost when TF-IDF restricted to this pre-selected set!
  - > considerable speed up

Results (20bit)



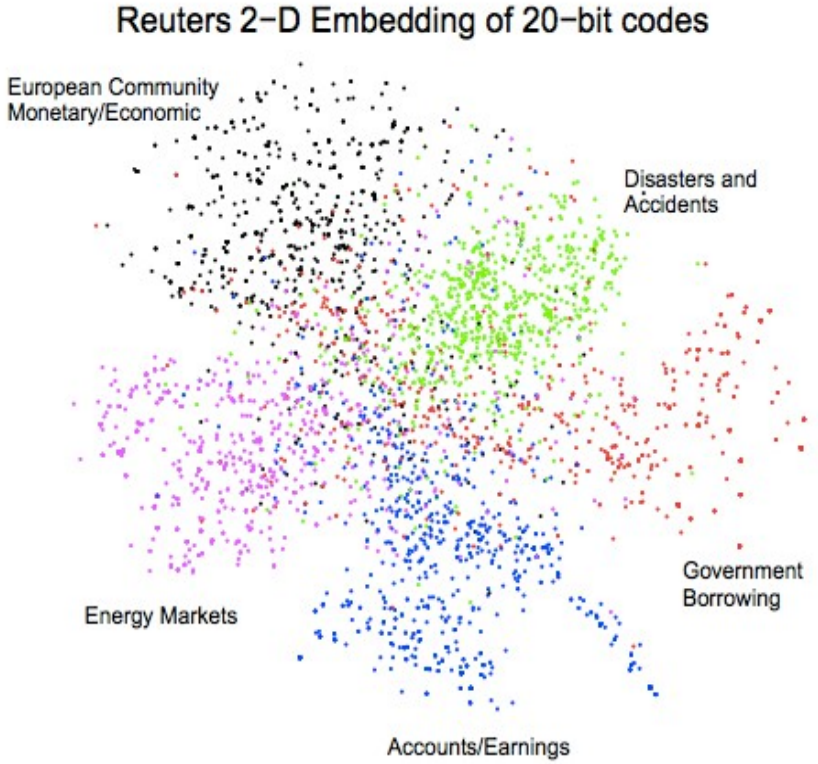


Figure 8: Left panel: Precision-Recall curves for the Reuters RCV2 dataset, when a query document from the test set is used to retrieve other test set documents, averaged over all 402,207 possible queries. Right panel: 2-dimensional embedding of the 20-bit codes using stochastic neighbor embedding for the Reuters RCV2 corpus. See in color for better visualization.

Some Numbers

- 30-bit for 1 billion docs: 1 doc/address; requires a few Gbs of memory
- Hamming Ball radius 5 → 175000 shortlist w/no search (can simply enumerate when required)
- Scaling learning is not difficult
  - Training on 10^9 docs takes < few weeks with 100 cores
  - "large organization" could train on many billions
- No need to generalize to new data if learning is ongoing (should improve upon results)

Potential problem

- Documents with similar addresses have similar content, but converse is not necessarily true
- Could have multiple spread out regions which are the same internally and also same externally, but far apart.
- Potential fix: add an extra penalty term during optimization → can use information about relevance of documents to construct this term
- → can backpropagate this through the net

How to View Semantic Hashing

- Each of the binary values in the code represents a set containing about half the document collection
- We want to intersect these sets for particular features
- Semantic hashing is a way of mapping set intersections required directly onto address bus
- Address bus can intersect sets with a single machine instruction!

Overview of Deep Learning NLP

- Colorful variety of approaches
- Started a while ago, revival of old ideas today applied to more data and better systems
- → Neural Net Language Model (Bengio)
- → RNNLM (use recurrent instead of feedforward)
- Skip-gram (2013) (simplification good)
- Paragraph Vector (2014) (beats Socher)
- LSTMs for MT (2014) (Sequence – Sequence w/LSTM)
- Semantic Hashing (Autoencoders)
- We did not cover: → Socher and RNTN for instance

Thank you for listening!

Citations

Note: some of the papers on here were used for reference and understanding purposes – not all were presented

1. Vincent, P. & Bengio, Y. A Neural Probabilistic Language Model. 3, 1137–1155 (2003).

2. Mikolov, T., Karafi, M. & Cernock, J. H. A Recurrent Neural Network Based Language Model. 1045–1048 (2010).

3. Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O. & Zaremba, W. Addressing the Rare Word Problem in Neural Machine Translation. 1–11 (2014). at <http://arxiv.org/abs/1410.8206>

4. Le, Q., Mikolov, T. & Com, T. G. Distributed Representations of Sentences and Documents. 32, (2014).

5. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Distributed Representations of Words and Phrases and their Compositionality. 1–9 (2013).

6. Mikolov, T., Chen, K., Corrado, G. & Dean, J. Efficient Estimation of Word Representations in Vector Space. 1–12 (2013). at <http://arxiv.org/abs/1301.3781>

7. Morin, F. & Bengio, Y. Hierarchical Probabilistic Neural Network Language Model.

8. Grauman, K. & Fergus, R. Learning Binary Hash Codes for Large-Scale Image Search.

9. Smith, N. A. Log-Linear Models. 1–9 (2004).

10. Krogh, A. Neural Network Ensembles , Cross Validation , and Active Learning.

11. Gutmann, M. Noise-contrastive estimation : A new estimation principle for unnormalized statistical models. 297–304 (2009).

12. Salakhutdinov, R. & Hinton, G. Semantic hashing. Int. J. Approx. Reason. 50, 969–978 (2009).

13. Sutskever, I., Vinyals, O. & Le, Q. V. Sequence to Sequence Learning with Neural Networks. 9 (2014). at <http://arxiv.org/abs/1409.3215>

14. Goldberg, Y. & Levy, O. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. 1–5 (2014). at <http://arxiv.org/abs/1402.3722>