Natural Language Processing Using Machine Learning

Boonserm Kijsirikul

Dept. of Computer Engineering

Chulalongkorn University

What is Natural Language Processing

- To process or understand natural language in order to perform tasks that are useful, e.g.
  - ➢ Question answering
  - ➢ Making appointment
  - ➢ Machine Translation
- NLP is a field at the intersection of
  - ➢ Computer Science
  - ➢ Artificial Intelligence
  - ➢ Linguistics

Applications of NLP

- Spell checking, keyword search
- Extracting information from websites such as
  - product price, dates, location
- Classifying: positive/negative sentiment of documents
- Machine translation
- Spoken dialog system
- Chatbots

Why is NLP hard?

- Complexity in representing, learning and using linguistic/situational/world/visual knowledge
- Human languages are ambiguous
- Human language interpretation depends on real world, common sense, and contextual knowledge

Word Embedding

- Word embedding maps a word from its original hign-dimensional input-space to a lower-dimensional vector space; embedding the word in a different space.
  - ➢Similarity in meaning ←→ similarity in vectors
  - ➢The context where a word appear gives meaning to it
  - ➢Use big datasets
- Word embedding becomes a fundamental component to many NLP tasks, e.g. sentiment analysis, machine translation, named-entity recognition.
- Word embedding techniques: word2vec, GloVe, fastText

Word Representation

- One-hot representation

  e.g. school = [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]

  university = [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
- Problems
  - ➢ no relationship between synonym words

    e.g. adept, expert, good, practiced, proficient, skillful
  - ➢ missing new words (cannot keep up to date)

    cool, awesome, fun, boss, fizzing, eximious
  - ➢ difficult to compute word similarity,

    e.g. vectors of “school” and “university” are orthogonal

    $[0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0]^T[0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\,0\,0] = 0$

You shall know the word by the company it keeps (J. R. Firth 1957)

- Language is symbolic in nature
  - ➢Surface form is in an arbitrary relation with the meaning of the word
  - ➢Hat vs. cat
    - One substitution: Levenshtein distance of 1
    - Does not measure the semantic similarity of the words
- Distributional semantics
  - ➢Linguistic items which appear in similar contexts in large samples of language data have similar meanings

Distributional Similarity Based Representations

- Example:
  - ➢I love having cereal in morning for breakfast.
  - ➢My breakfast is usually jam with butter.
  - ➢The best part of my day is morning’s fresh coffee with a hot breakfast.
- ‘cereal’, ‘jam’, ‘butter’, and ‘coffee’ are related.
- We need to represent each word such that similar words have similar representation.

Distributional Similarity Based Representations

government debt problems turning into banking crises as has happened in

saying that Europe needs unified banking regulation to replace the hodgepodge

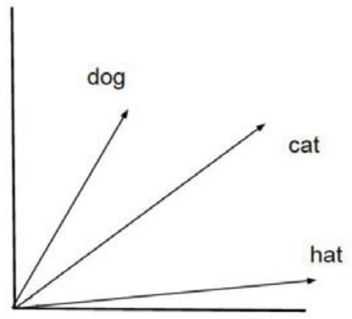
↖ These words will represent *banking* ↗

- Build a dense vector for each word, chosen so that it is good at predicting other words appearing in its context

$$\text{banking} = \begin{bmatrix} 0.125 \\ -0.285 \\ -0.314 \\ 0.112 \\ 0.881 \\ -0.453 \\ 0.613 \end{bmatrix}$$

Word Vectors

- Words represented by their context as vectors
- Proximity in vector space indicates semantic or functional similarity

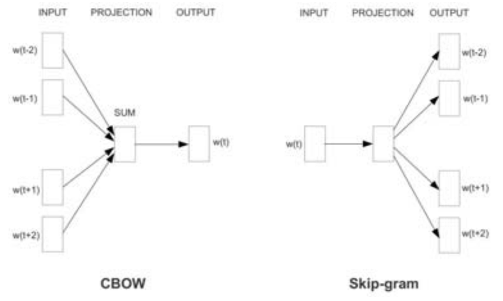


Vector similarity such as cosine

$$1 - \frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_i\|_2 \|\mathbf{x}_j\|_2}$$

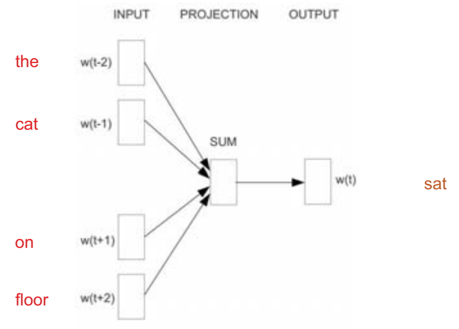
Represent the meaning of word – word2vec

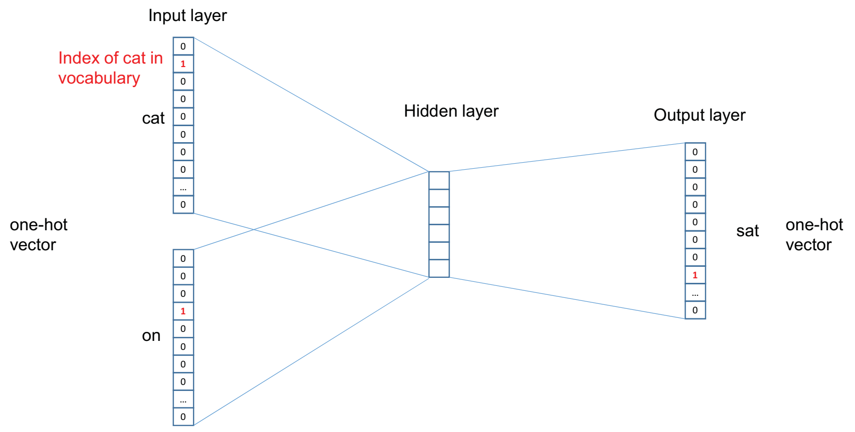
- 2 basic neural network models:
  - Continuous Bag of Word (CBOW): use a window of word to predict the middle word
  - Skip-gram (SG): use a word to predict the surrounding ones in window.

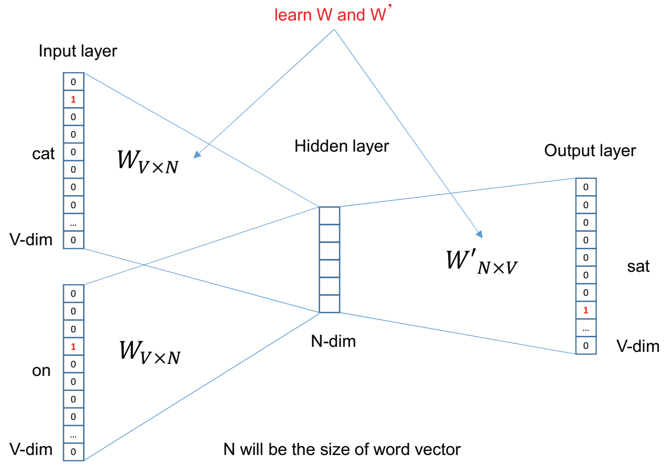


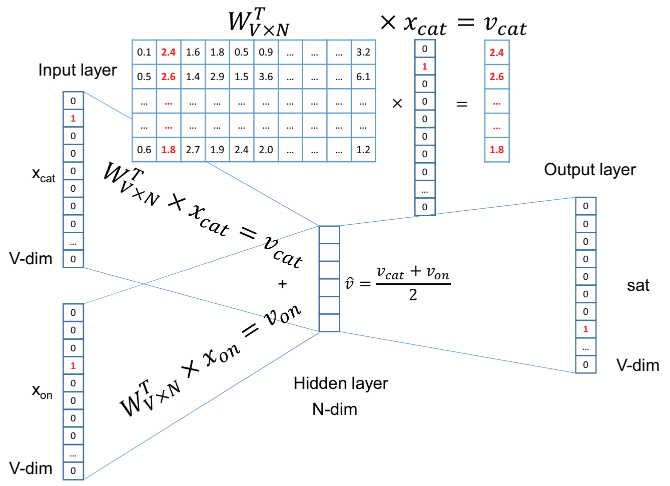
Word2vec – Continuous Bag of Word

- E.g. "The cat sat on floor"
  - Window size = 2



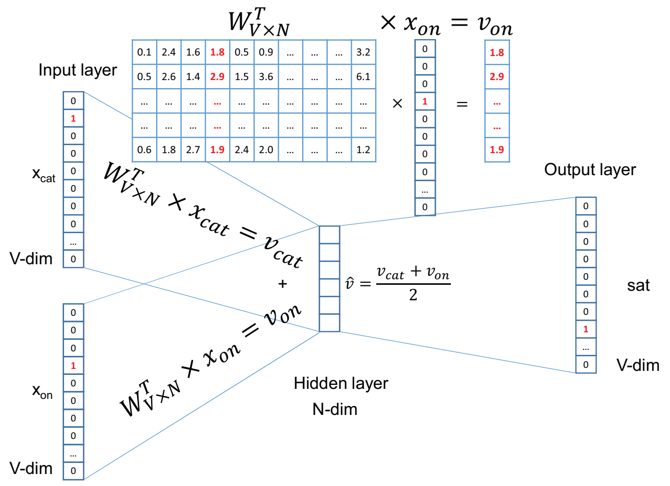





$$W^T_{V\times N} \times x_{cat} = v_{cat}$$

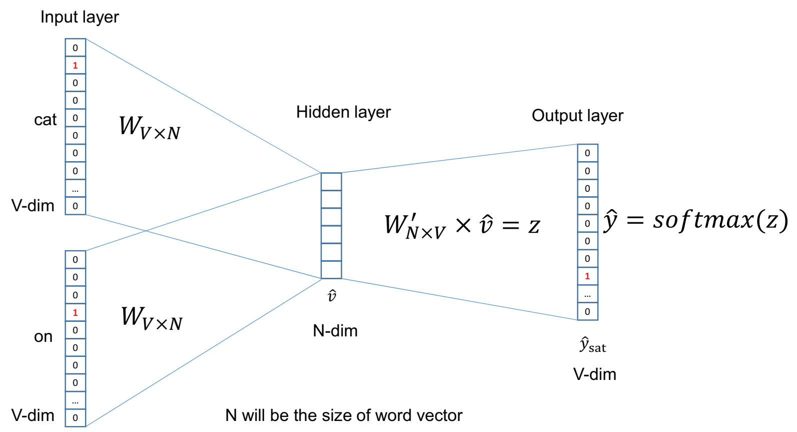
$$\begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 2.4 \\ 2.6 \\ \dots \\ \dots \\ 1.8 \end{bmatrix}$$

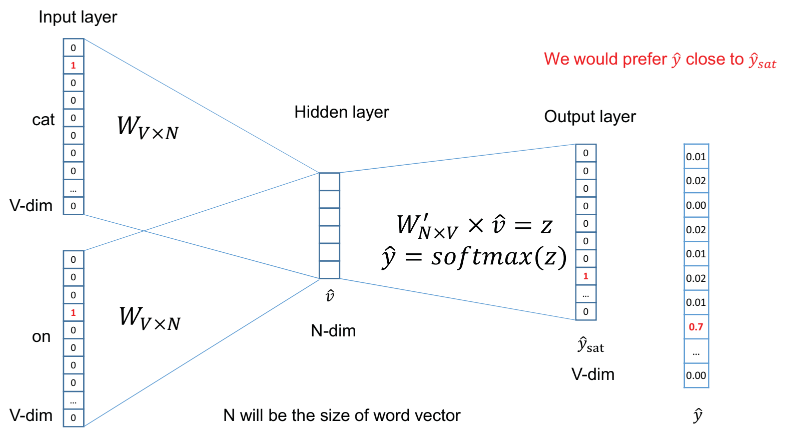
Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim

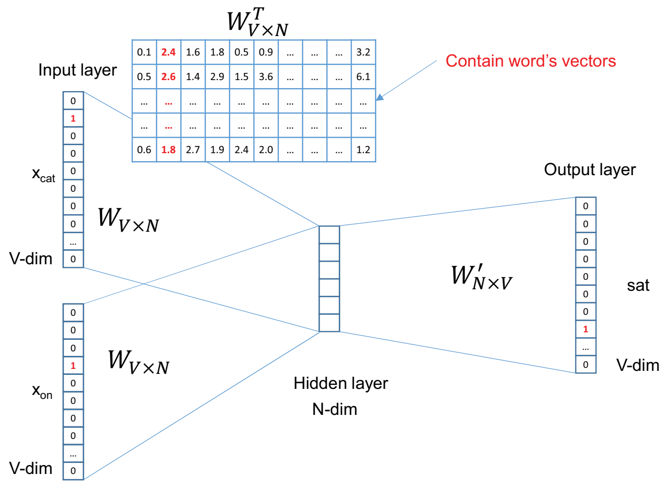

$$W^T_{V\times N} \times x_{on} = v_{on}$$

$$\begin{bmatrix} 0.1 & 2.4 & 1.6 & 1.8 & 0.5 & 0.9 & \dots & \dots & \dots & 3.2 \\ 0.5 & 2.6 & 1.4 & 2.9 & 1.5 & 3.6 & \dots & \dots & \dots & 6.1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0.6 & 1.8 & 2.7 & 1.9 & 2.4 & 2.0 & \dots & \dots & \dots & 1.2 \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ \dots \\ 0 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.9 \\ \dots \\ \dots \\ 1.9 \end{bmatrix}$$

Input layer
x_{cat}
V-dim
x_{on}
V-dim
$W^T_{V\times N} \times x_{cat} = v_{cat}$
+
$W^T_{V\times N} \times x_{on} = v_{on}$
$\hat{v} = \frac{v_{cat} + v_{on}}{2}$
Hidden layer
N-dim
Output layer
sat
V-dim



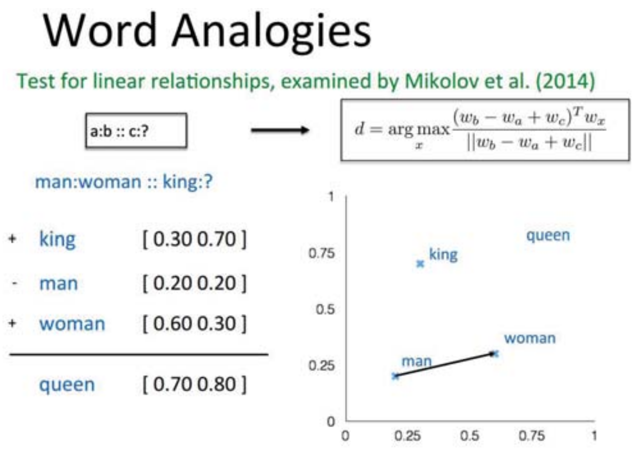




0.1	2.4	1.6	1.8	0.5	0.9	...	...	...	3.2
0.5	2.6	1.4	2.9	1.5	3.6	...	...	...	6.1
...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...
0.6	1.8	2.7	1.9	2.4	2.0	...	...	...	1.2

We can consider either W or W' as the word's representation.
Or even take the average.

Some interesting results



Word Analogies

Test for linear relationships, examined by Mikolov et al. (2014)

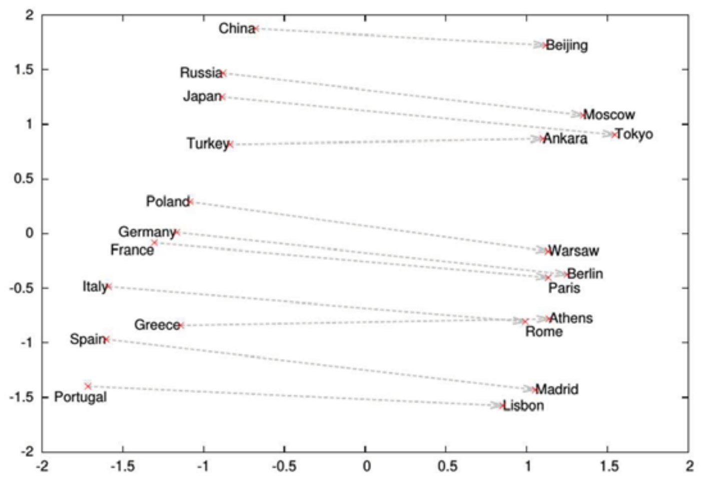
a:b :: c:? $\longrightarrow$ $d = \arg\max_{x} \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$

man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

1
0.75
0.5
0.25
0
0
0.25
0.5
0.75
1
king
queen
woman
man

Word Analogies



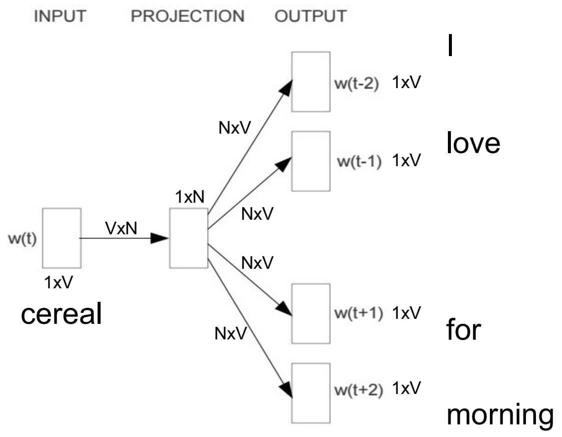
Word2Vec: Skip-Gram

- *Insurgents killed in ongoing fighting.*
- **Bi-grams** = {insurgents killed, killed in, in ongoing, ongoing fighting}
- **2-skip-bi-grams** = {insurgents killed, insurgents in, insurgents ongoing, killed in, killed ongoing, killed fighting, in ongoing, in fighting, ongoing fighting}
- **Tri-grams** = {insurgents killed in, killed in ongoing, in ongoing fighting}
- **2-skip-tri-grams** = {insurgents killed in, insurgents killed ongoing, insurgents killed fighting, insurgents in ongoing, insurgents in fighting, insurgents ongoing fighting, killed in ongoing, killed in fighting, killed ongoing fighting, in ongoing fighting}.

Word2Vec: Skip-Gram

- Neural network trained on context of each word.
- Predicts the context given a word.
- The predicted context is used as the feature representation of a particular word in a review.
- We need to combine the feature vectors of the words in a review to get the overall feature vector of the review.

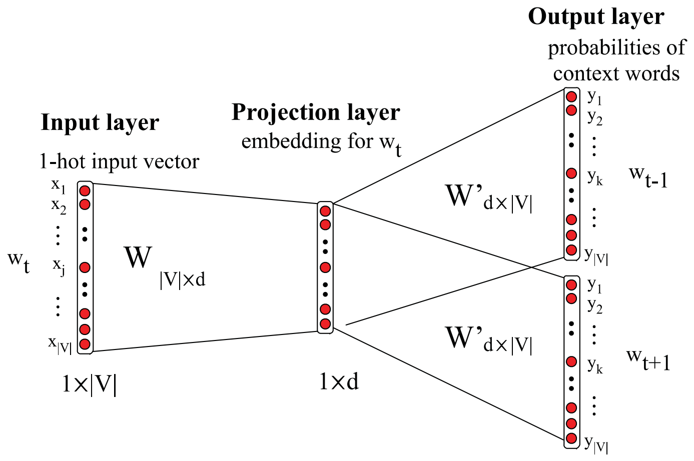
Word2Vec: Skip-gram



V = #distinct words

Network Structure

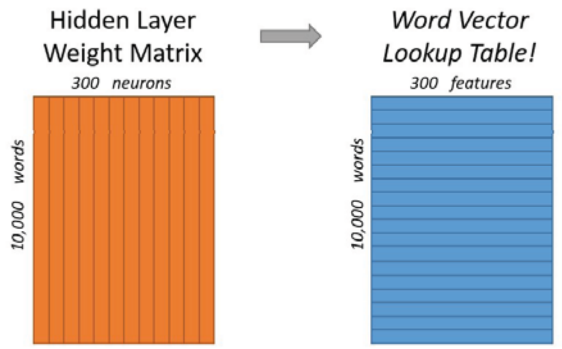
- Assume the vocabulary of 10,000 unique words. (V=10,000)
- Represent an input word like "ants" as a one-hot vector
- Assume that the word vector has 300 features
- The output of the network is a single vector (also with 10,000 components) for each context word
- Assume that the number of context words is equal to 2



Slide courtesy of Jurafsky & Martin

Hidden Layer

- Assume that the word vector has 300 features
  → hidden layer is a weight matrix with
    - ➢10,000 rows (one for every word in vocabulary)
    - ➢300 columns (one for every hidden neuron).



Language Models

- A language model assigns a probability to a sequence of words, such that $\sum_{w\in\Sigma^*} p(w) = 1$
- Given the observed training text, how probable is a new utterance?
- Thus we compare different orderings of words (e.g. translation)

  p(he likes apples) > p(apple likes he)

  or choice of words (e.g. speech recognition)

  p(he likes apples) > p(he licks apple)

From https://github.com/oxford-cs-deepnlp-2017/lectures

Language Models

- Much of Natural Language Processing can be structured as (conditional) language modeling:
  - ➢ Translation

    p_{LM}(Les chiens aiment les os ||| Dogs love bones)
  - ➢ Question Answering

    p_{LM}(What do dogs love? ||| bones)
  - ➢ Dialogue

    p_{LM}(How are you? ||| Fine thanks. And you?)

Language Models

- Most language models employ the chain rule to decompose the joint probability into a sequence of conditional probabilities:

$$p(w_1, w_2 \dots w_N) =$$
$$p(w_1)p(w_2|w_1) \times\cdots\times p(w_N|w_1, w_2, \dots, w_{N-1})$$

Note that this decomposition is exact and allows us to model complex joint distributions by learning conditional distributions over the next word (w_n) given the history of words observed ($w_1,\dots,w_{n\text{-}1}$).

Language Models

- The simple objective of modeling the next word given the observed history contains much of the complexity of natural language understanding.
- Consider predicting the extension of the utterance:

$p(\cdot|$ There she built a)

- With more context we are able to use our knowledge of both language and the world to heavily constrain the distribution over the next word:

$p(\cdot|$ Alice went to the beach. There she built a)

There is evidence that human language acquisition partly relies on future prediction.

Evaluating a Language Model

- A good model assigns real utterances w_1^N from a language a high probability. This can be measured with cross entropy:

$$H(w_1^N) = -\frac{1}{N}\log_2 p(w_1^N)$$

- Intuition 1: Cross entropy is a measure of how many bits are needed to encode text with our model.
- Alternatively we can use perplexity:

$$\text{perplexity}(w_1^N) = 2^{H(w_1^N)}$$

- Intuition 2: Perplexity is a measure of how surprised our model is on seeing each word.

Language Modeling Data

- Language modeling is a time series prediction problem in which we must be careful to train on the past and test on the future.
- If the corpus is composed of articles, it is best to ensure the test data is drawn from a disjoint set of articles to the training data.

Language Modeling Data

- Two popular data sets for language modelling evaluation are a pre-processed version of the Penn Treebank (www.fit.vutbr.cz/~imikolov/rnnlm/simple -examples.tgz), and the Billion Word Corpus (code.google.com/p/1-billion-word-language-modeling-benchmark/).
- The PTB is very small and has been heavily processed. As such it is not representative of natural language.
- The Billion Word corpus was extracted by first randomly permuting sentences in news articles and then splitting into training and test sets. As such train and test sentences come from the same articles and overlap in time.

Three Approaches

- Three approaches to parameterising language models:
  - ➢ With count based n-gram models we approximate the history of observed words with just the previous n words.
  - ➢ Neural n-gram models embed the same fixed n-gram history in a continues space and thus better capture correlations between histories.
  - ➢ With Recurrent Neural Networks we drop the fixed n-gram history and compress the entire history in a fixed length vector, enabling long range correlations to be captured.

(1) Count based N-Gram Language Models

Markov assumption:

- only previous history matters
- limited memory: only last k - 1 words are included in history (older words less relevant)
- kth order Markov model

For instance 2-gram language model:

$$p(w_1, w_2, w_3 \dots, w_n) \\ = p(w_1)p(w_2|w_1)p(w_3|w_1, w_2) \times \cdots \\ \times p(w_N|w_1,\ w_2, \dots, w_{n-1}) \\ \approx p(w_1)p(w_2|w_1)p(w_3|w_2) \times \cdots \times p(w_N|w_{N-1})$$

- The conditioning context, $w_{i\text{-}1}$, is called the history.

Estimating Probabilities

Maximum likelihood estimation for 3-grams:

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

Collect counts over a large text corpus. Billions to trillions of words are easily available by scraping the web.

Back-Off

In our training corpus we may never observe the trigrams:

- Oxford Pimm's eater
- Oxford Pimm's drinker

If both have count 0 our smoothing methods will assign the same probability to them.

A better solution is to interpolate with the bigram probability:

- Pimm's eater
- Pimm's drinker

Interpolated Back-Off

By recursively interpolating the n-gram probabilities with the (n - 1)-gram probabilities we can smooth our language model and ensure all words have non-zero probability in a given context.

A simple approach is linear interpolation:

$$p(w_n|w_{n-2},w_{n-1}) =$$
$$\lambda_3 p(w_n|w_{n-2},w_{n-1}) + \lambda_2 p(w_n|w_{n-1}) + \lambda_1 p(w_n)$$

- where $\lambda_3 + \lambda_2 + \lambda_1 = 1$

Provisional Summary

Good

- Count based n-gram models are exceptionally scalable and are able to be trained on trillions of words of data,
- fast constant time evaluation of probabilities at test time,
- sophisticated smoothing techniques match the empirical distribution of language.

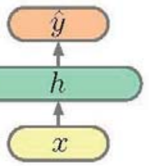
Bad

- Large n-grams are sparse, so hard to capture long dependencies,
- symbolic nature does not capture correlations between semantically similar word distributions, e.g. cat ←→ dog,
- similarly morphological regularities, running ←→ jumping, or gender.

(2) Neural N-Gram Language Models

- Feed forward network

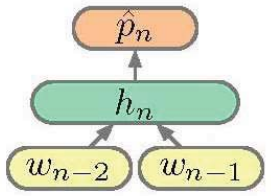
$$h = g(Vx + c)$$
$$\hat{y} = Wh + b$$



Trigram NN Language Model

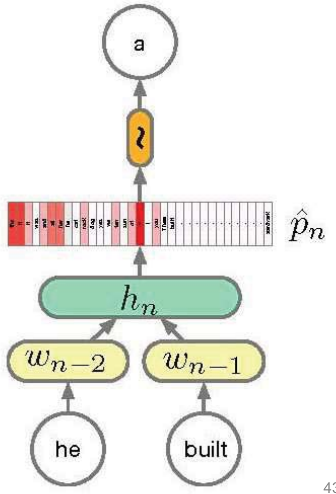
$$h_n = g(V[W_{n-1}; W_{n-2}] + c)$$
$$\hat{p}_n = softmax(Wh_n + b)$$
$$softmax(u)_i = \frac{\exp(u_i)}{\sum_j \exp(u_j)}$$

- W_i are one hot vectors and $\hat{p}_i$ are distributions,
- $|W_i| = |\hat{p}_i| = V$ (words in the vocabulary),
- V is usually very large > 1e5.



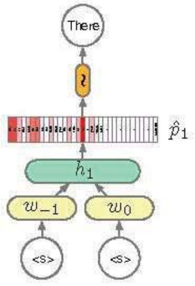
Sampling

$w_n | w_{n-1}; w_{n-2} \approx \hat{p}_n$



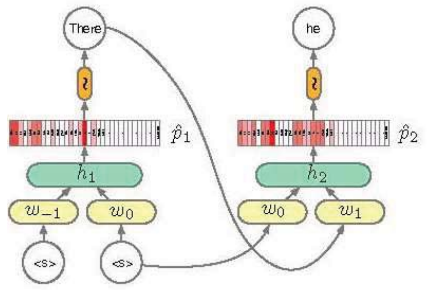
Sampling

$w_n | w_{n-1}; w_{n-2} \approx \hat{p}_n$



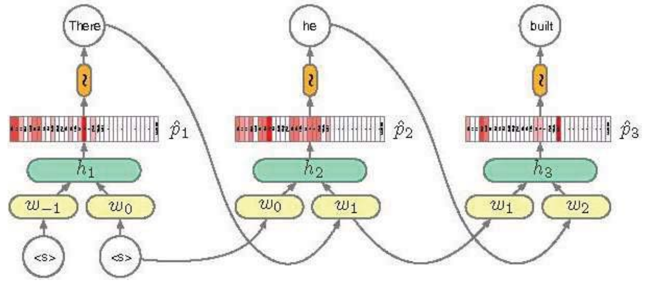
Sampling

$w_n | w_{n-1}; w_{n-2} \approx \hat{p}_n$



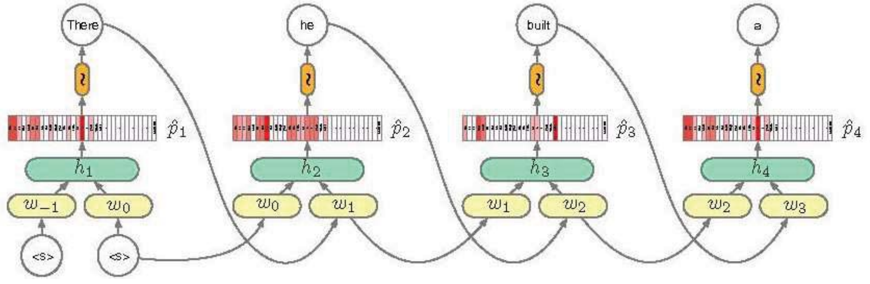
Sampling

$w_n | w_{n-1}; w_{n-2} \approx \hat{p}_n$



Sampling

$w_n | w_{n-1}; w_{n-2} \approx \hat{p}_n$

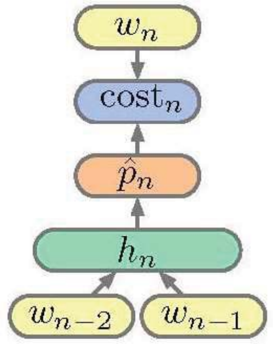


Training

The usual training objective is the cross entropy of the data given the model (MLE):

$$\mathcal{F} = -\frac{1}{N}\sum_n \text{cost}_n(w_n, \hat{p}_n)$$

The cost function is simply the model's estimated log-probability of w_n: $cost(a; b) = a^T \log(b)$ (assuming w_i is a one hot encoding of the word)

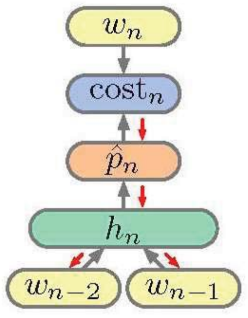


Training

Calculating the gradients is straightforward with back propagation:

$$\frac{\partial \mathcal{F}}{\partial W} = -\frac{1}{N}\sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial W}$$

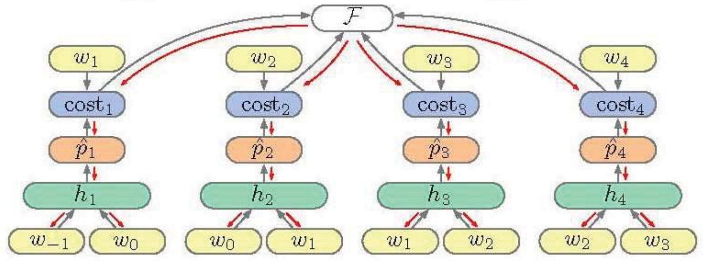
$$\frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{N}\sum_n \frac{\partial \text{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$



Training

Calculating the gradients is straightforward with back propagation:

$$\frac{\partial \mathcal{F}}{\partial W} = -\frac{1}{4}\sum_{n=1}^{4} \frac{\partial \text{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial W} \quad , \quad \frac{\partial \mathcal{F}}{\partial V} = -\frac{1}{4}\sum_{n=1}^{4} \frac{\partial \text{cost}_n}{\partial \hat{p}_n}\frac{\partial \hat{p}_n}{\partial h_n}\frac{\partial h_n}{\partial V}$$



Note that calculating the gradients for each time step n is independent of all other timesteps, as such they are calculated in parallel and summed.

Comparison with Count Based N-Gram LMs

Good

- Better generalisation on unseen n-grams, poorer on seen n-grams. Solution: direct (linear) ngram features.
- Simple NLMs are often an order magnitude smaller in memory footprint than their vanilla n-gram cousins (though not if you use the linear features suggested above!).

Bad

- The number of parameters in the model scales with the n-gram size and thus the length of the history captured.
- The n-gram history is finite and thus there is a limit on the longest dependencies that can be captured.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.

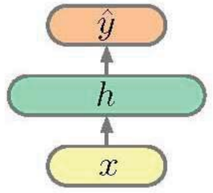
(3) Recurrent Neural Network Language Models

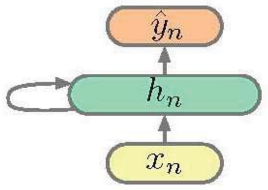
- Feed forward network

$$h = g(Vx + c)$$
$$\hat{y} = Wh + b$$

- Recurrent network

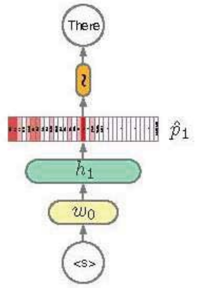
$$h_n = g(V[x_n; h_{n-1}] + c)$$
$$\hat{y}_n = Wh_n + b$$





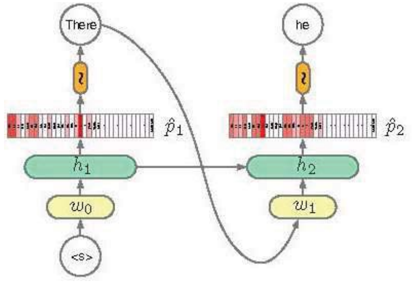
Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



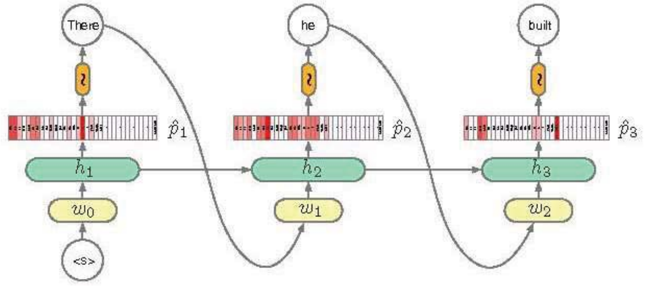
Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



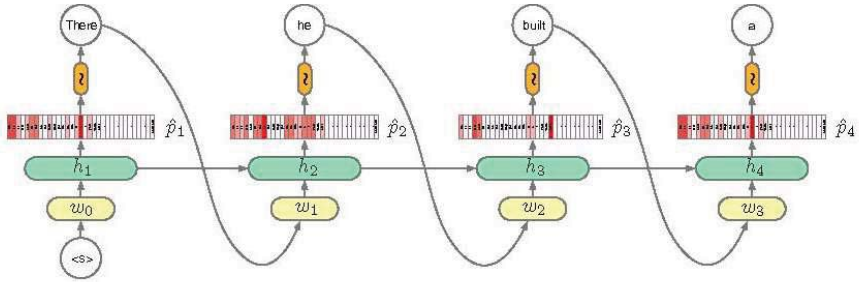
Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



Recurrent Neural Network Language Models

$$h_n = g(V[x_n; h_{n-1}] + c)$$



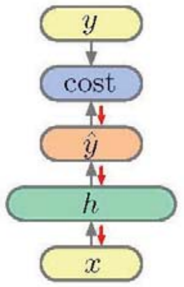
Recurrent Neural Network Language Models

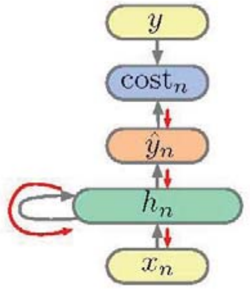
- Feed forward network

$$h = g(Vx + c)$$
$$\widehat{y} = Wh + b$$

- Recurrent network

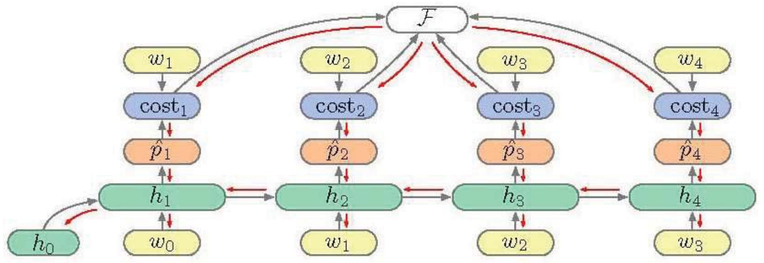
$$h_n = g(V[x_n; h_{n-1}] + c)$$
$$\hat{y}_n = Wh_n + b$$





Recurrent Neural Network Language Models

- The unrolled recurrent network is a directed acyclic computation graph. We can run backpropagation as usual:

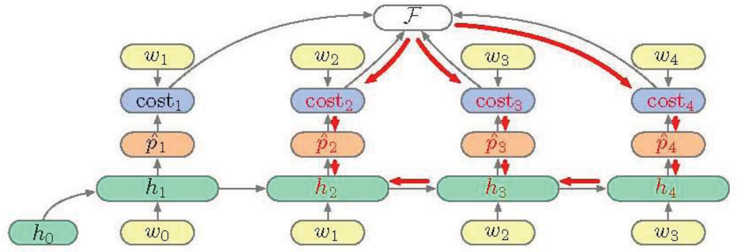


Recurrent Neural Network Language Models

This algorithm is called Back Propagation Through Time (BPTT).

Note the dependence of derivatives at time *n* with those at time *n* + *α*

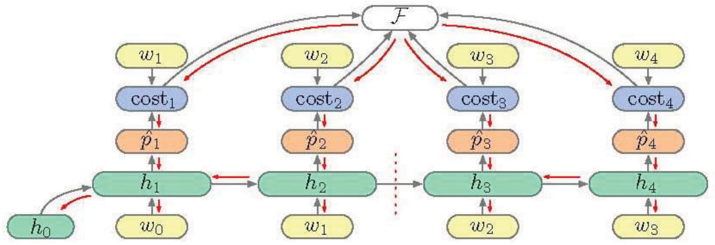
$$\frac{\partial \mathcal{F}}{\partial h_2} = \frac{\partial \mathcal{F}}{\partial \text{cost}_2}\frac{\partial \text{cost}_2}{\partial \hat{p}_2}\frac{\partial \hat{p}_2}{\partial h_2} + \frac{\partial \mathcal{F}}{\partial \text{cost}_3}\frac{\partial \text{cost}_3}{\partial \hat{p}_3}\frac{\partial \hat{p}_3}{\partial h_3}\frac{\partial h_3}{\partial h_2} + \frac{\partial \mathcal{F}}{\partial \text{cost}_4}\frac{\partial \text{cost}_4}{\partial \hat{p}_4}\frac{\partial \hat{p}_4}{\partial h_4}\frac{\partial h_4}{\partial h_3}\frac{\partial h_3}{\partial h_2}$$



Recurrent Neural Network Language Models

If we break these depdencies after a fixed number of timesteps we get Truncated Back Propagation Through Time (TBPTT):

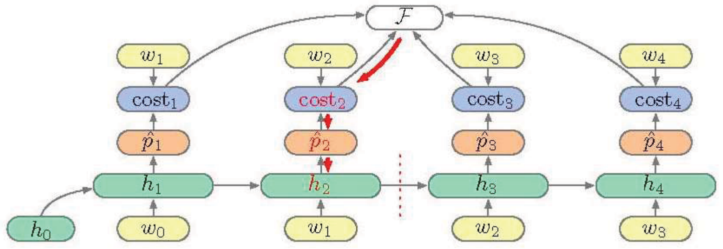
$$\mathcal{F} = -\frac{1}{4}\sum_{n=1}^{4} \text{cost}_n(w_n, \hat{p}_n)$$



Recurrent Neural Network Language Models

If we break these depdencies after a fixed number of timesteps we get Truncated Back Propagation Through Time (TBPTT):

$$\frac{\partial \mathcal{F}}{\partial h_2} \approx \frac{\partial \mathcal{F}}{\partial \text{cost}_2}\frac{\partial \text{cost}_2}{\partial \hat{p}_2}\frac{\partial \hat{p}_2}{\partial h_2}$$



Comparison with N-Gram LMs

Good

- RNNs can represent unbounded dependencies, unlike models with a fixed n-gram order.
- RNNs compress histories of words into a fixed size hidden vector.
- The number of parameters does not grow with the length of dependencies captured, but they do grow with the amount of information stored in the hidden layer.

Bad

- RNNs are hard to learn and often will not discover long range dependencies present in the data.
- Increasing the size of the hidden layer, and thus memory, increases the computation and memory quadratically.
- Mostly trained with Maximum Likelihood based objectives which do not encode the expected frequencies of words a priori.