Christof Monz

Deep Learning

Language Models and Word Embeddings

Today's Class

- N-gram language modeling
- Feed-forward neural language model
  - Architecture
  - Final layer computations
- Word embeddings
  - Continuous bag-of-words model
  - Skip-gram
  - Negative sampling

The Role of LM in SMT

- Translation models map source phrases to target phrases
  - Translation probabilities should reflect the degree to which the meaning of the source phrase is preserved by the target phrase (adequacy)
  - source: "Der Mann hat einen Hund gekauft."
    monotone translation: "The man has a dog bought."
    Translation preserves the meaning but is not fluent

The Role of LM in SMT

- Translation models map source phrases to target phrases
  - Translation probabilities should reflect the degree to which the meaning of the source phrase is preserved by the target phrase (adequacy)
  - source: "Der Mann hat einen Hund gekauft."
    monotone translation: "The man has a dog bought."
    Translation preserves the meaning but is not fluent
- Language models compute the probability of a string
  - $p(\text{the man has a dog bought.}) < p(\text{the man has bought a dog.})$
  - Language model probabilities do not necessarily correlate with grammaticality: $p(\text{green ideas sleep furiously.})$ is likely to be small
  - During translation language model scores of translation hypotheses are compared to each other

The Role of LM in SMT

- The language model is one of the most important models in SMT
- Substantial improvements in translation quality can be gained from carefully trained language models
- Decades of research (and engineering) in language modeling for Automated Speech Recognition (ASR)
  - Many insights can be transferred to SMT
  - Types of causes for disfluencies differ between both areas
    ASR: $p(\text{We won't I scream}) < p(\text{We want ice cream})$
    SMT: $p(\text{Get we ice cream}) < p(\text{We want ice cream})$
  - Reordering does not play a role in ASR

N-gram Language Modeling

- N-gram language model compute the probability of a string as the product of probabilities of the consecutive n-grams:
  - $p(\text{<s> the man has a dog bought . </s>})$
    $= p(\text{<s> the}) \cdot p(\text{<s> the man}) \cdot p(\text{the man has}) \cdot p(\text{man has a}) \cdot p(\text{has a dog}) \cdot p(\text{a dog bought}) \cdot p(\text{dog bought .}) \cdot p(\text{bought . </s>})$

N-gram Language Modeling

- N-gram language model compute the probability of a string as the product of probabilities of the consecutive n-grams:
  - $p(\text{<s> the man has a dog bought . </s>})$ $= p(\text{<s> the}) \cdot p(\text{<s> the man}) \cdot p(\text{the man has}) \cdot p(\text{man has a}) \cdot p(\text{has a dog}) \cdot p(\text{a dog bought}) \cdot p(\text{dog bought .}) \cdot p(\text{bought . </s>})$
  - Generally: $p(w_1^N) = \prod_{i=1}^{N} p(w_i | w_{i-n+1}^{i-1})$, for order n
  - Problem: if one n-gram probability is zero, e.g., $p(\text{dog bought .}) = 0$, then the probability of the entire product is zero
  - Solution: smoothing

Language Model Smoothing

- A number of smoothing approaches have been developed for language modeling
- Jelinek-Mercer smoothing
  - Weighted linear interpolation of conditional probabilities of different orders
- Katz smoothing
  - Back-off to lower-order probabilities and counts are discounted
- Witten-Bell smoothing
  - Linear interpolation where lower-order probabilities are weighted by the number of contexts of the history
- Kneser-Ney smoothing
  - Weight lower-order probabilities by the number of contexts in which they occur

Kneser-Ney Smoothing

$$p_{\mathrm{KN}}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} \frac{\max\{c(w_{i-n+1}^{i}) - D(c(w_{i-n+1}^{i})),0\}}{\sum_{w_i} c(w_{i-n+1}^{i})} & \text{if } c(w_{i-n+1}^{i}) > 0 \\ \gamma(w_{i-n+1}^{i-1}) p_{\mathrm{KN}}(w_i|w_{i-n+2}^{i-1}) & \text{if } c(w_{i-n+1}^{i}) = 0 \end{cases}$$

- Original backoff-style formulation of Kneser-Ney smoothing
  - Closer to representation found in ARPA style language models
  - Can be re-formulated as linear interpolation (see Chen and Goodman 1999)

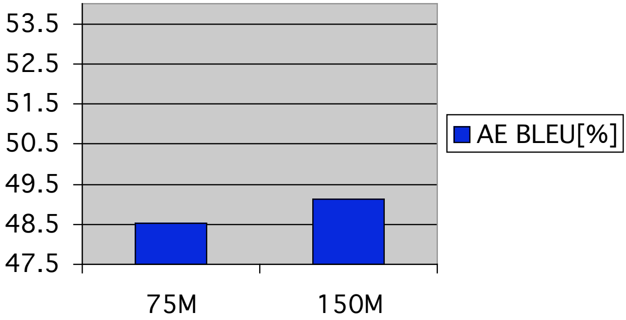
LM Smoothing in SMT

- Does the choice of smoothing method matter for SMT?
  - Kneser-Ney smoothing typically yields results with the lowest perplexity
  - Correlation between perplexity and MT metrics (such a BLEU) is low
  - Few comparative studies, but Kneser-Ney smoothing yields small gains over Witten-Bell smoothing
- Kneser-Ney smoothing is the de facto standard for SMT (and ASR)
- Recent SMT research combines Witten-Bell smoothing with Kneser-Ney smoothing

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:

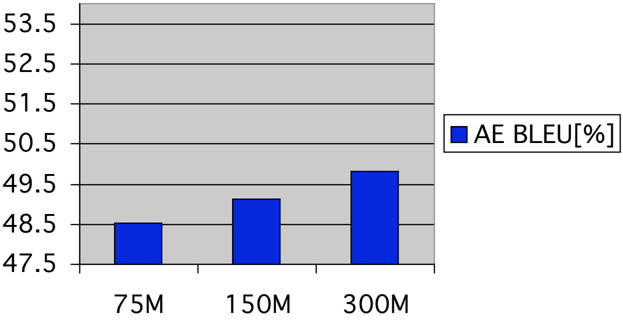


9

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:

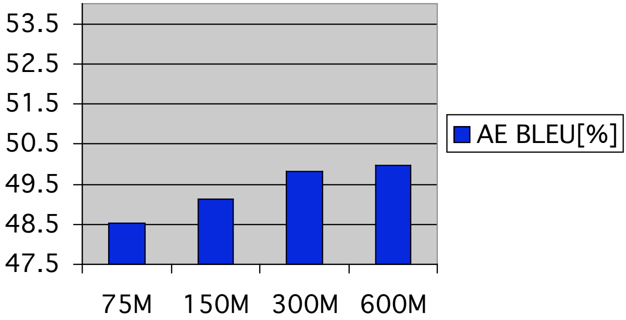


10

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:

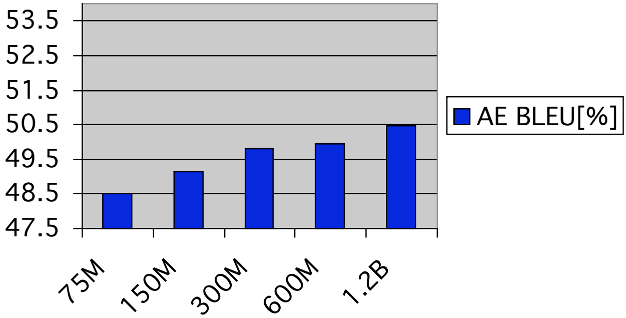


11

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:

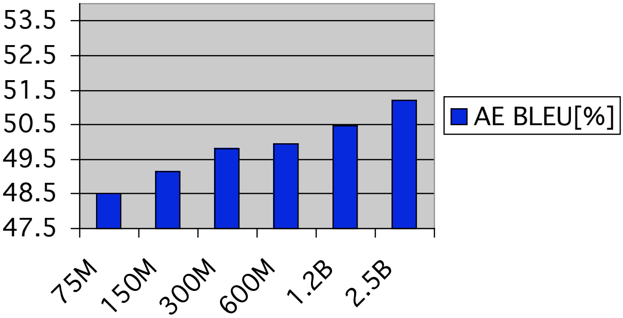


12

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



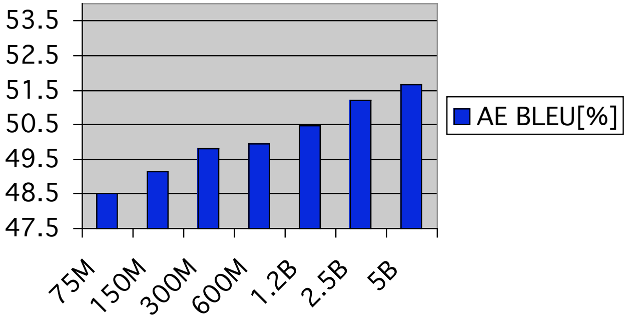
13

Google

Size Matters

More data is better data…

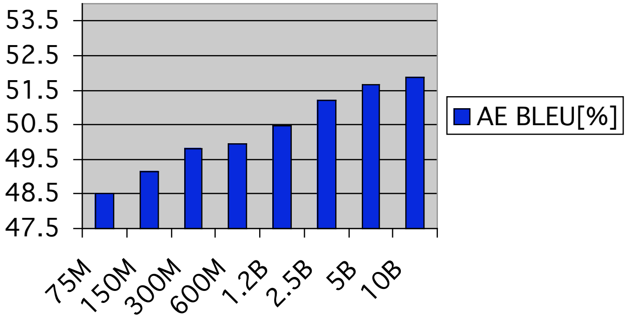
Five-gram language model, no count-cutoff, integrated into search:



Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:



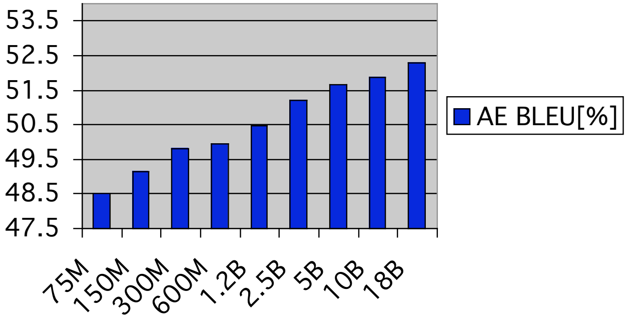
15

Google

Size Matters

More data is better data…

Five-gram language model, no count-cutoff, integrated into search:

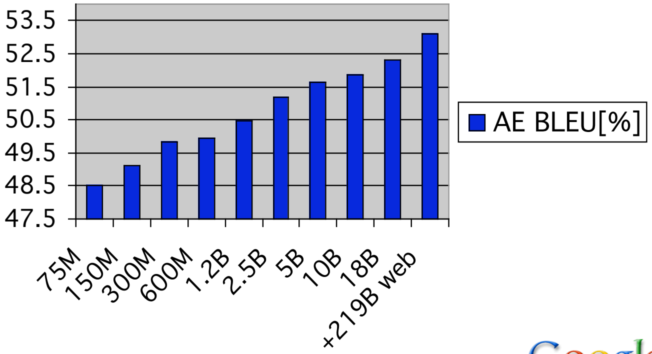


16

Size Matters

More data is better data…

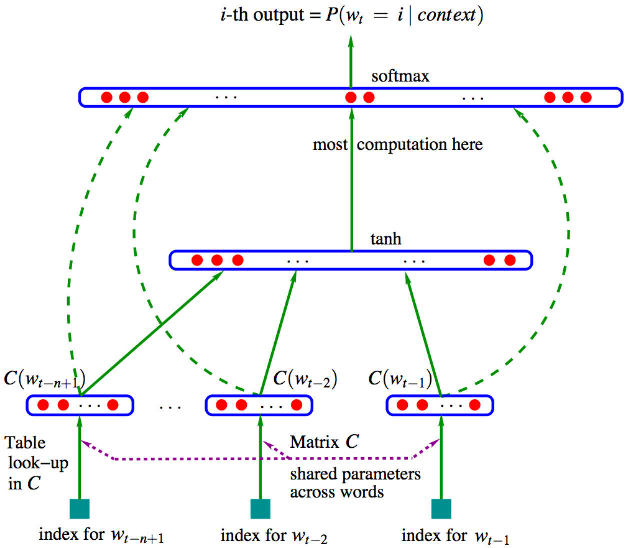
Five-gram language model, no count-cutoff, integrated into search:



Probabilistic Neural Network LMs

- Both word- and class-based models use discrete parameters as elements of the event space
- The current word+history n-gram has not been seen during training or it has not been seen (binary decision)
  - Smoothing results in a more relaxed matching criterion
- Probabilistic Neural Network LMs (Bengio et al. JMLR 2003) use a distributed real-valued representation of words and contexts
- Each word in the vocabulary is mapped to a m-dimensional real-valued vector
  - $C(w) \in \mathbb{R}^m$, typical values for m are 50, 100, 150
  - A hidden layer capture the contextual dependencies between words in an n-gram
  - The output layer is a $|V|$-dimensional vector describing the probability distribution of $p(w_i|w_{i-n+1}^{i-1})$

Probabilistic Neural Network LMs



Probabilistic Neural Network LMs

- Layer-1 (projection layer)

  $$C(w_{t-i}) = Cw_{t-i}$$

  where

  - w_{t-i} is a V-dimensional 1-hot vector, i.e., a zero-vector where only the index corresponding the word occurring at position $t-i$ is 1
  - C is a $m \times V$ matrix

- Layer-2 (context layer)

  $$h = \tanh(d + Hx)$$

  where

  - $x = [C(w_{t-n+1});\ldots;Cw_{t-1}]$ $\quad$ ($[\cdot\,;\,\cdot]$ = vector concatenation)
  - H is a $n \times (l-1)m$ matrix

Probabilistic Neural Network LMs

- Layer-3 (output layer)

  $\hat{y} = \text{softmax}(b + U\,h)$

  where

  - U is a $V \times n$ matrix
  - $\text{softmax}(v) = \frac{\exp(v_i)}{\sum_i \exp(v_i)}$ (turns activations into probs)

- Optional: skip-layer connections

  $\hat{y} = \text{softmax}(b + W\,x + U\,h)$

  where

  - W is a $V \times (l-1)m$ matrix (skipping the non-linear context layer)

Training PNLMs

- Loss function is cross-entropy: $L(y, \hat{y}) = -\log(\hat{y}_i)$, where $i = \text{argmax}(y)$
- Optimize with respect to $\frac{\partial L(y,\hat{y})}{\partial \theta}$
  where $\theta = \{C, H, d, U, b\}$ using stochastic gradient descent (SGD)

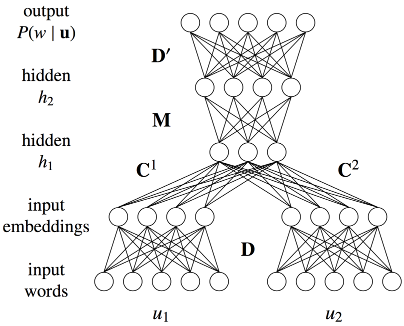
Training PNLMs

- Loss function is cross-entropy: $L(y, \hat{y}) = -\log(\hat{y}_i)$, where $i = \text{argmax}(y)$
- Optimize with respect to $\frac{\partial L(y,\hat{y})}{\partial \theta}$

  where $\theta = \{C, H, d, U, b\}$ using stochastic gradient descent (SGD)
- Update all parameters, including C (the projections)
- What does C capture?
  - maps discrete words to continuous, low dimensional vectors
  - C is shared across all contexts
  - C is position-independent
  - if $C(white) \approx C(red)$ then $p(drives|a\ white\ car) \approx p(drives|a\ red\ car)$

PNLM Variant

- Previous architecture directly connects hidden context layer to full vocabulary output layer
- Alternative: introduce output projection layer in between:



- Sometimes also referred to as 'deep output layer'

How useful are PNLMs?

- Advantages:
  - PNLMs outperform n-gram based language models (in terms of perplexity)
  - Use limited amount of memory
    - NPLM: $\sim$100M floats $\approx$ 400M RAM
    - n-gram model: $\sim$10-40G RAM
- Disadvantages:
  - Computationally expensive
    - Mostly due to large output layer (size of vocabulary): Uh can involve hundreds of millions of operations!
    - We want to know $p(w|C)$ for a specific w, but to do so we need softmax over entire output layer

Speeding up PNLMs

- Slow training
  - annoys developpers/scientists/PhD students
  - slows down development cycles
- Slow inference
  - annoys users
  - can cause products to become impractical

Speeding up PNLMs

- Slow training
  - annoys developpers/scientists/PhD students
  - slows down development cycles
- Slow inference
  - annoys users
  - can cause products to become impractical
- Speeding things up
  - Mini-batching (training)
  - Using GPUs (training)
  - Parallelization (training)
  - Short-lists (training + inference)
  - Class-based structured output layers (training + inference)
  - Hierarchical softmax (training + inference)
  - Noise contrastive estimation (training + inference)
  - Self-normalization (inference)

Mini-Batching

- Instead of computing $p(w|C)$ compute $p(W|\mathcal{C})$ where W is an ordered set of words, and $\mathcal{C}$ is ordered set of contexts
- $\Rightarrow$ Matrix-matrix multiplications instead of matrix-vector multiplications
  allows to use low-level libraries such as BLAS to exploit memory-layout
- $\hat{y} = \mathrm{softmax}(b + U \tanh(d + Hx))$ becomes
  $\hat{Y} = \mathrm{softmax}(b + U \tanh(d + HX))$
- Advantage: Mini-batching is very GPU friendly
- Disadvantage: fewer parameter updates (depends on mini-batch size)
- Disadvantage: not really applicable during inference

Short-lists

- In NLP, the size of the vocabulary can easily reach 200K (English) to 1M (Russian) words
- Quick-fix: short-lists
  - ignore rare words and keep only the n most frequent words
  - all rare words are mapped to a special token: <unk>
- Typical sizes of short-lists vary between 10K, 50K, 100K, and sometimes 200K words
- Disadvantage: all rare words receive equal probability (in a given context)

Class-Based Output Layer

- Partition vocabulary into n non-overlapping classes (C)
  - using clustering (Brown clustering)
  - fixed categories (POS tags)
- Instead of $\hat{y} = \text{softmax}(b + U h)$

  compute $\hat{c} = \text{softmax}(b + U h)$, where $|c| \ll |V|$

  then choose $\hat{c}_i = \text{argmax}(\hat{c})$ and

  compute $\hat{y}_{c_i} = \text{softmax}(b + U_{c_i} h)$

  where U_{c_i} is a $|V_{c_i}| \times |h|$ matrix, where $|V_{c_i}| \ll |V|$

Class-Based Output Layer

- Partition vocabulary into n non-overlapping classes (C)
  - using clustering (Brown clustering)
  - fixed categories (POS tags)
- Instead of $\hat{y} = \text{softmax}(b + U h)$

  compute $\hat{c} = \text{softmax}(b + U h)$, where $|c| \ll |V|$

  then choose $\hat{c}_i = \text{argmax}(\hat{c})$ and

  compute $\hat{y}_{c_i} = \text{softmax}(b + U_{c_i} h)$

  where U_{c_i} is a $|V_{c_i}| \times |h|$ matrix, where $|V_{c_i}| \ll |V|$
- Advantage: leads to significant speed improvements
- Disadvantage: not very mini-batch friendly (matrix U_{c_i} can vary across instances in the same batch)

Self-Normalization

- During inference (i.e., when applying a trained model to unseen data) we are interested in $p(w|c)$ and not $p(w'|c)$, where $w' \neq w$
- Unfortunately $b + Uh$ does not yield probabilities and softmax requires summation over the entire output layer
- 'Encourage' the neural network to produce probability-like values (Devlin et al., ACL-2014) without applying softmax

Self-Normalization

- Softmax log likelihood:

  $$\log(P(x)) = \log(\frac{\exp(U_r(x))}{Z(x)})$$

  where

  - $U_r(x)$ is the output layer score for x
  - $Z(x) = \sum_{r'=1}^{|V|} U_{r'}(x)$

  $$\log(P(x)) = \log(U_r(x)) - \log(Z(x))$$

- If we could ensure that $\log(Z(x)) = 0$ then we could use $\log(U_r(x))$ directly
- Strictly speaking not possible, but we can encourage the model by augmenting the loss function:

  $$L = \sum_i [\log(P(x_i)) - \alpha(\log(Z(x_i))^2]$$

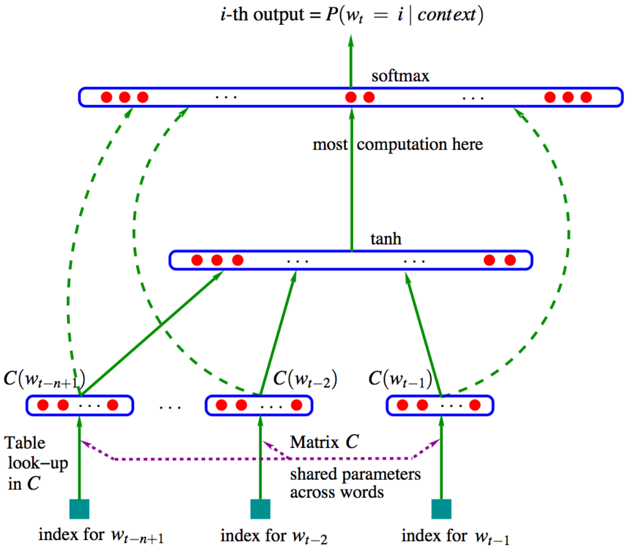
Self-Normalization

- Self-normalization included during training; for inference, $\log(P(x)) = \log(U_r(x))$
- α regulates the importance of normalization (hyper-parameter):

Arabic BOLT Val		
α	$\log(P(x))$	$\lvert\log(Z(x))\rvert$
0	-1.82	5.02
10^{-2}	-1.81	1.35
10^{-1}	-1.83	0.68
1	-1.91	0.28

- Initialize output layer bias to $\log(1/|V|)$
- Devlin et al. report speed-ups of around 15x during inference
- No speed-up during training

Reminder: PNLM Architecture



Projections = Embeddings?

- Are projections the same as word embeddings?
- What are (good) word embeddings? $C(w) \approx C(w')$ iff
  - w and w' mean the same thing
  - w and w' exhibit the same syntactic behavior
- For PNLMs the projections/embeddings are by-products
  - Main objective is to optimize next word prediction
  - Projections are fine-tuned to achieve this objective
- Representation learning: if the main objective is to learn good projections/embeddings

Word Meanings

- What does a word mean?
- Often defined in terms of relationship between words
  - Synonyms: `purchase :: acquire` (same meaning)
  - Hyponyms: `car :: vehicle` (is-a)
  - Meronyms: `wheel :: car` (part-whole)
  - Antonyms: `small :: large` (opposites)
- Explicit, qualitative relations require hand-crafted resources (dictionaries, such as WordNet)
  - expensive
  - incomplete
  - language-specific
- What about
  - learning relations automatically?
  - quantifying relations between words, e.g., $\mathrm{sim}(car, vehicle) > \mathrm{sim}(car, tree)$?

Distributional Semantics

- "*You shall know a word by the company it keeps.*" (Firth, 1957)

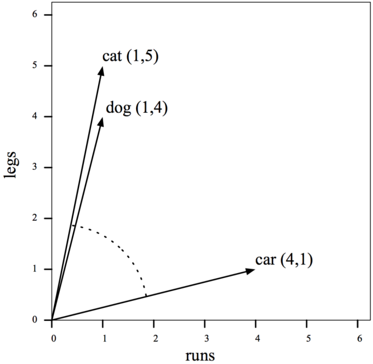
word	context vector					
	leash	walk	run	owner	pet	bark
dog	3	5	2	5	3	2
cat	0	3	3	2	3	0
lion	0	3	2	0	1	0
light	0	0	0	0	0	0
bark	1	0	0	2	1	0
car	0	0	1	3	0	0

- In distributional semantics all words w are represented as a V-dimensional context vector c_w
- $c_w[i] = f$ where f is the frequency of word i occurring within the (fixed-size) context of w

Distributional Semantics

- Word similarity as cosine similarity in the context vector space:

word	context vector	
	runs	legs
dog	1	4
cat	1	5
car	4	1



- In distributional semantics context vectors are high-dimensional, discrete, and sparse

Word Embeddings

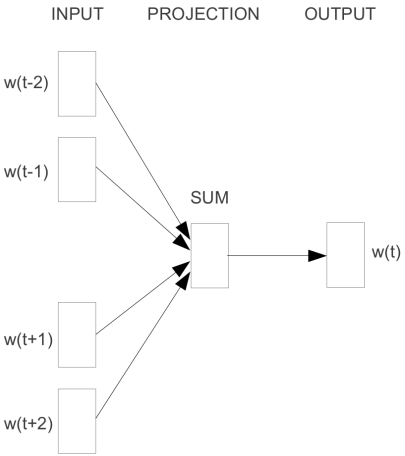
- Similar underlying intuition to distributional semantics, but word vectors are
  - low dimensional (e.g., 100 vs. $|V|$)
  - dense (no zeros)
  - continuous ($c_w \in \mathbb{R}^m$)
  - learned by performing a task (predict)
- Popular approach: Word2Vec (Mikolov et al.)
- Word2Vec consists of two approaches:
  - Continuous Bag of Words (CBOW)
  - Skip-Gram

Continuous Bag of Words (CBOW)

- Task: Given a position t in a sentence, and the n words occurring to its the left ($\{w_{t-n}, \ldots, w_{t-1}\}$) and m its right ($\{w_{t+1}, \ldots, w_{t+n}\}$) predict the word in position t

  the man X *the road*, with $X = ?$
- Seemingly similar to n-gram language modeling where $n =$ LM order -1 and $m = 0$
- Use feed-forward neural network
  - Focus on learning embeddings themselves
  - Simpler network (compared to PNLM)
  - Bring embedding/projection layer closer to output
  - Typically $n = m$, and $n \in \{2, 5, 10\}$

CBOW Model Architecture



CBOW Model

- No non-linearities
- One hidden layer:

  $h = \frac{1}{2n} W w_C$, where

  - W is a $|h| \times |V|$ matrix
  - $w_C = \sum_{i=t-n, i \neq t}^{t+n} w_i$
  - w_i is a 1-hot vector for the word occurring in position i
- Output layer:

  $\hat{y} = \mathrm{softmax}(W' h)$

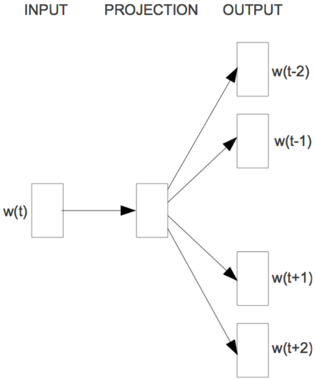
  - W' is a $|V| \times |h|$ matrix
  - W' and W are not (necessarily) shared, i.e., $W' \neq W^T$
- Loss function: cross entropy (see PNLM)
- Trained with SGD

CBOW Embeddings

- Where do the embeddings live?
  - Column i in W ($|h| \times |V|$ matrix) represents the embedding for word i
  - Row i in W' ($|V| \times |h|$ matrix) represents the embedding for word i
- Which one of the two?
  - Typically W or
  - $W_s = W^T + W'$ (combining both into one)

Skip-Gram Model Architecture

- Alternative to CBOW
- Task: Given a word at position t in a sentence, predict the words occurring between positions $t-n$ and $t-1$ and between $t+1$ and $t+n$



Skip-Gram Model

- One hidden layer:

  $h = W w_I$, where

  - w_I is the 1-hot vector for word at position t

- $2n$ output layers:

  $p(w_{t-n} \ldots w_{t-1} w_{t+1} \ldots w_{t+n} | w_I)$

  $\propto \prod_{i=t-n, i \neq t}^{t+n} p(w_i | w_I)$

  $\hat{y}_i = \text{softmax}(W' h)$ ($t - n \leq i \leq t + n$ and $i \neq t$)

  - W' is a $|V| \times |h|$ matrix
  - W' and W are not (necessarily) shared, i.e., $W' \neq W^T$

- Loss function: cross entropy (see PNLM)
- Trained with SGD

Negative Sampling

- Both CBOW and Skip-gram benefit from large amounts of data
- Computing activations for the full output layer becomes an issue
- Negative sampling: Try to distinguish between words that do and and words that do not occur in the context of the input word
  - Classification task
  - 1 positive example (from the ground truth)
  - k negative examples (from a random noise distribution

Negative Sampling

- Given the input word w and a context word c we want to

$$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

where D represents the observed data and D' a noise distribution

Negative Sampling

- Given the input word w and a context word c we want to

  $$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

  where D represents the observed data and D' a noise distribution
- We compute $p(D=1|c,w;\theta)$ as $\sigma(v_c \cdot v_w)$

  where $v_w = W\,w$ and $v_c = W'^T\,c$

Negative Sampling

- Given the input word w and a context word c we want to

  $$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

  where D represents the observed data and D' a noise distribution
- We compute $p(D=1|c,w;\theta)$ as $\sigma(v_c \cdot v_w)$

  where $v_w = W\,w$ and $v_c = W'^T\,c$
- $p(D=0|c,w;\theta) = 1 - p(D=1|c,w;\theta)$

Negative Sampling

- Given the input word w and a context word c we want to

  $$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

  where D represents the observed data and D' a noise distribution
- We compute $p(D=1|c,w;\theta)$ as $\sigma(v_c \cdot v_w)$

  where $v_w = W w$ and $v_c = W'^T c$
- $p(D=0|c,w;\theta) = 1 - p(D=1|c,w;\theta)$
- Since $1 - \sigma(x) = \sigma(-x)$:

  $$\arg\max_{\theta} \prod_{(w,c)\in D} \sigma(v_c \cdot v_w) \prod_{(w,c)\in D'} \sigma(-v_c \cdot v_w)$$

Negative Sampling

- Given the input word w and a context word c we want to

  $$\arg\max_{\theta} \prod_{(w,c)\in D} p(D=1|c,w;\theta) \prod_{(w,c)\in D'} p(D=0|c,w;\theta)$$

  where D represents the observed data and D' a noise distribution
- We compute $p(D=1|c,w;\theta)$ as $\sigma(v_c \cdot v_w)$

  where $v_w = W w$ and $v_c = W'^T c$
- $p(D=0|c,w;\theta) = 1 - p(D=1|c,w;\theta)$
- Since $1-\sigma(x) = \sigma(-x)$:

  $$\arg\max_{\theta} \prod_{(w,c)\in D} \sigma(v_c \cdot v_w) \prod_{(w,c)\in D'} \sigma(-v_c \cdot v_w)$$

  $$\arg\max_{\theta} \sum_{(w,c)\in D} \log\sigma(v_c \cdot v_w) + \sum_{(w,c)\in D'} \log\sigma(-v_c \cdot v_w)$$

Word2Vec Practical Considerations

- Skip-Gram:
  - For each observer occurrence (w, c) add 5-20 negative samples to data
  - Draw c from uni-gram distribution $P(w)$
  - Scale uni-gram distribution: $P(w)^{0.75}$ to bias rarer words
- Context size typically around 2-5
- The more data the smaller the context and the negative sample set
- Exclude very rare words (less than 10 occurrences)
- Removing stop words: better topical modeling, less sensitive to syntactical patterns

Evaluation of Word Embeddings

- Word similarity tasks
  - Rank list of word pairs, e.g., (*car*,*bicycle*), by similarity
  - Spearman correlation with human judgements
  - Benchmarks: WS-353, Simlex-999, ...
  - Mixes all kinds of similarities (synonyms, topical, unrelated...)
- Analogy task
  - Paris is to France as Berlin is to X
  - Evaluated by accuracy
  - Also includes syntactic analogy: *acquired* is to *acquire* as *tried* is to X
  - Arithmetic magic: $X = v_{king} - v_{man} + v_{woman}$

Applicability of Word Embeddings

- Word similarity
- To initialize projection layers in deep networks
  - if training data is small
  - if number of output classes is small
  - Task-specific fine-tuning still useful in many cases

Recap

- Feed-Forward Neural Language Model
  - Projection layers
  - Cross-entropy loss
  - Final layer computations
    - Mini-Batching
    - Short-lists
    - Class-based structured output layer
    - Self-normalization
- Word embeddings
  - Continuous bag-of-words model
  - Skip-gram
  - Negative sampling