WORD EMBEDDING

Pham Quang Khang

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

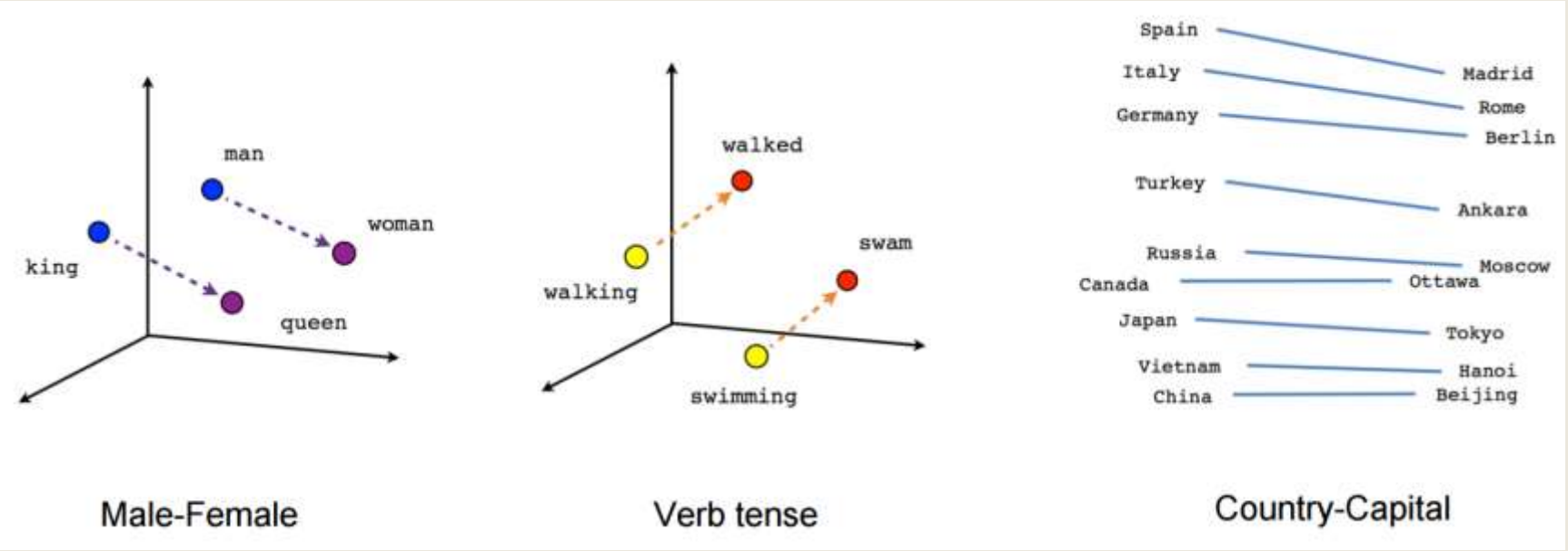
What is word embedding and why

- Word embedding: representation of words from the vocabulary in a "low"-dimensional vector space of real number.
  - Example: vocabulary size of 100,000 to vector space of dimension 300

- How about one hot vector?
  - Too computational expensive: hundreds thousand of vector with dimension of 100,000
  - Not possible to represent the relationship between words: synonym, tense, nuance
  - Difficult to represent the relationship between words within the context

What is the result of word embedding?

- A matrix M of dimension V x D: V is the number of words in the vocabulary, D is the designed dimension, D << V
- Words with similar meaning should stay "close" in the new vector space:
  - King – Man + Woman ~ Queen



https://www.tensorflow.org/tutorials/word2vec

Brief history of word embedding

- Count-based methods: Latent Semantic Analysis, Latent Semantic Indexing (using Singular Value Decomposition – SVD for reducing dimension of representation matrix)
- Predictive methods: neural probabilistic language models (Bengio et al. 2003): predict a word from its neighbors in terms of learned small, dense embedding vectors
  - 2003: Basic Neural Language model (Bengio et al.)
  - 2013: Word2vec (Mikolov et al.)
  - 2014: GloVe (Pennington et al.) (not in scope of this talk)
  - 2017: Fasttext (Facebook research group) -> extension of Word2Vec

Markov assumption – principle of language modeling

- From product rule: $p(\mathbf{x}_1, \ldots, \mathbf{x}_N) = \prod_{n=1}^{N} p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}).$
  - Too expensive and too complicated to calculate all probability

- Markov property : the conditional probability distribution of future states of the process depends only upon the present state
  - First- order $p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) = p(\mathbf{x}_n | \mathbf{x}_{n-1})$
  - k-order: $P(w_i \mid w_1 w_2 \ldots w_{i-1}) \approx P(w_i \mid w_{i-k} \ldots w_{i-1})$

- The problem became: compute the probability of a word w_i given its k previous words

$$P(w_i \mid w_{i-k} \ldots w_{i-1})$$

2 approaches in computing the probability

- Count base method:

  $P(w_t|w_{t-1}, \ldots, w_{t-k+1}) = count(w_t, w_{t-1}, \ldots, w_{t-k+1})/count(w_{t-1}, \ldots, w_{t-k+1})$

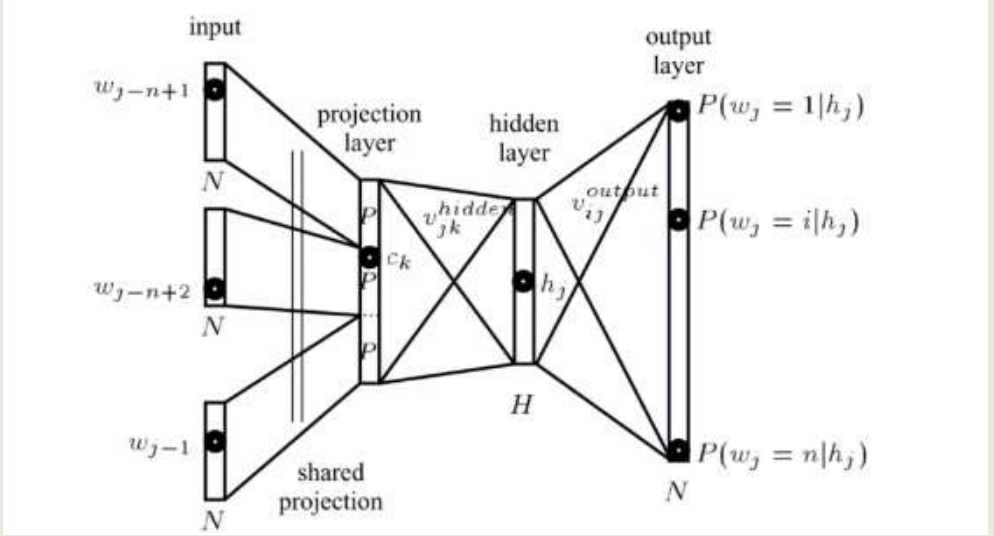
  setting for k = 5 and using with Kneser-Ney smoothing leads to smoothed 5-gram models that have been found to be a strong baseline (Ref 7)

- Neural network: softmax

$$p(w_t|w_{t-1}, \ldots, w_{t-k+1}) = \frac{\exp(h^T v'_t)}{\sum_{w_i} \exp(h^T v'_{w_i})}$$

Objective function

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} log\, p(w_t|w_{t-1}, \ldots, w_{t-k+1})$$



A neural language model (Bengio et al. 2006)

Bengio Classic Neural Language Model

- Model: one projection layer, one hidden layer:
- Output y, input w_i, projection C, hidden H

$$P(w_t|w_{t-1}, \ldots . w_{t-n+1}) = \frac{e^{y_{w_t}}}{\sum e^{y_i}}$$

$$y = b + Wx + Utanh(d + Hx)$$

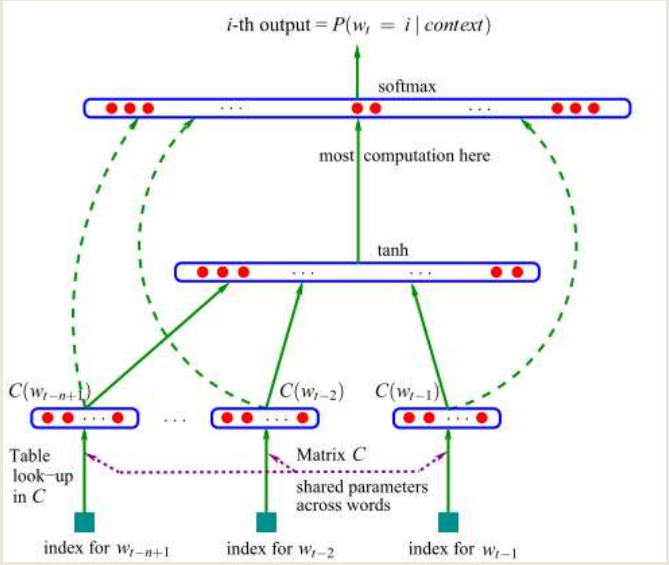
$$x = (C(w_{t-1}), C(w_{t-2}), \ldots C(w_{t-n+1}))$$

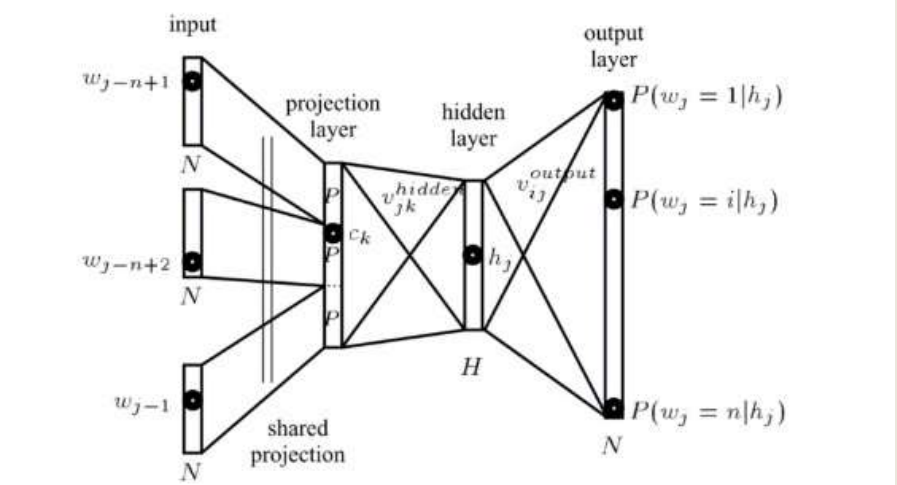
W: optional, can be zero: direct link from input to output

Matrix size:

C: (V, m), x: (nm, 1), W: (V, nm), U: (V, h), H: (h, nm)

- Most of computing load is on the softmax layer, which can be real problem with tens of thousands of word in vocabulary





Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Problem with Softmax in language modeling

- Probability of the output word w_t given context C (w_{t-1}, w_{t-2}, …, w_{t-n}):

$$f(w_t, w_{t-1}, w_{t-2}, \ldots, w_{t-n+1}) = \frac{e^{-g(w_t, w_{t-1}, w_{t-2}, \ldots, w_{t-n+1})}}{\sum_v e^{-g(v, w_{t-1}, w_{t-2}, \ldots, w_{t-n+1})}}$$

$$g(v, w_{t-1}, w_{t-2}, \ldots, w_{t-n+1}) = \text{a. tanh(c + Wx + UF}_v\text{)} + \text{b}$$

$$x = (F_{w_{t-1}}, F_{w_{t-2}}, \ldots, F_{w_{t-n+1}})'$$

F_{w_t} : Vector represents word at index t (by look-up from embedding matrix)

- Problem: calculating UF_v for all word within Vocabulary V in each training iter is too expensive (think of V as 10,000 or even 100,000)
- Objective of algorithm: reduce the computations for each step from O(V) to O(log(|V|)) for training phase
  - For inference, computing the probability of all words is necessary, hence just use softmax without further thinking

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Hierarchical Softmax – general idea

- Idea:
  - Focus on probability of one or small set of words (for training phase)
  - Partition the classes set and use the product rule to decomposition the expensive softmax

$$P(Y=y|X=x) = P(Y=y|C=c(y),X)\ P(C=c(y)|X=x)$$

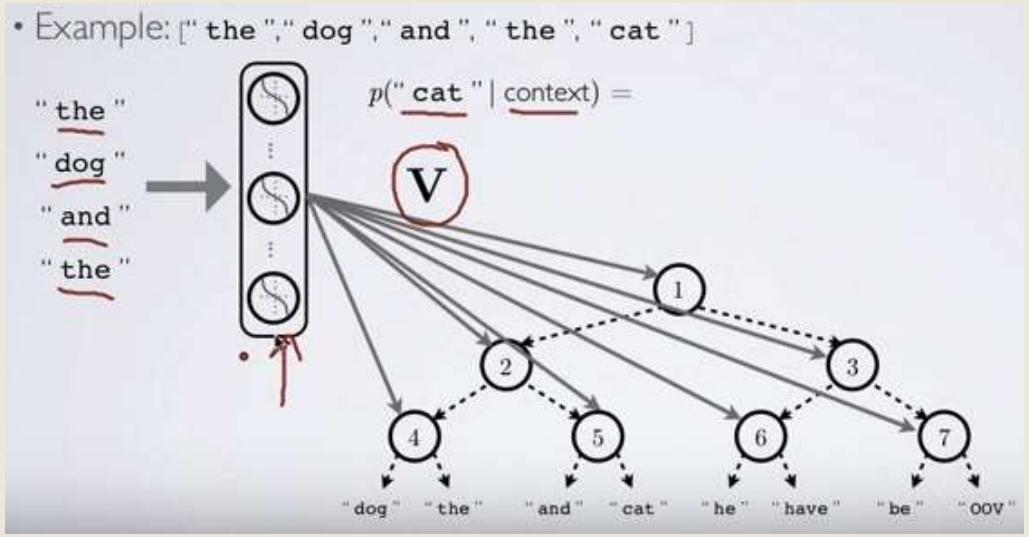
Why: $P(Y|X) = \Sigma\ P(Y, C=i|X) = \Sigma\ P(Y|C=i, X)\ P(C = i|X)$

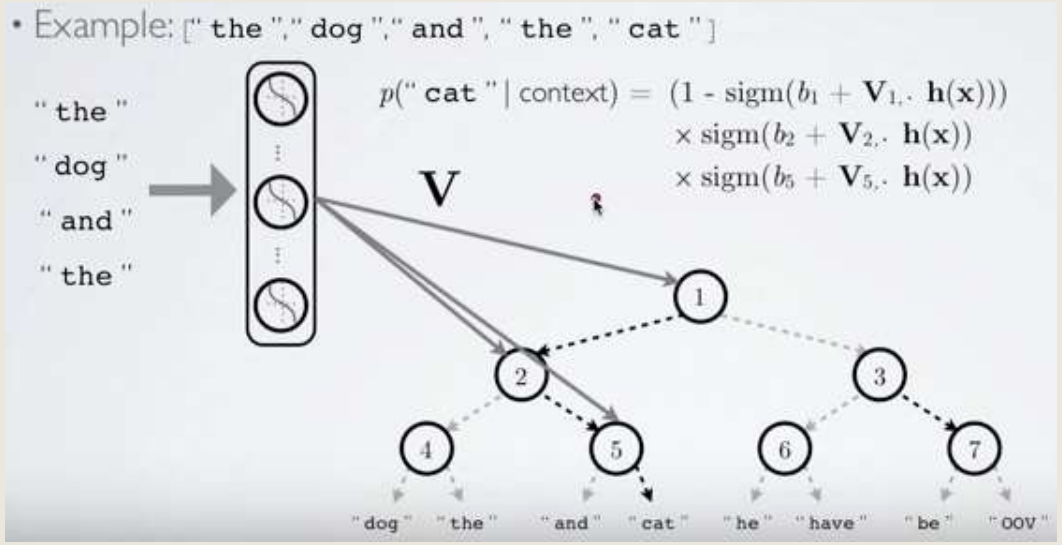
- More explanation:
  - Partition the classes of 10,000 into 100 classes of 100 with additional probability for word to go into subset as P(y in class Y_i) = α_i
  - The deeper the hierarchical to do partition for Vocabulary, the cheaper the computation is
  - Final probability can be represented as:

$$P(v|w_{t-1}, \dots, w_{t-n+1}) = \prod_{j=1}^{m} P(b_j(v)|b_1(v), \dots, b_{j-1}(v), w_{t-1}, \dots, w_{t-n+1})$$

Hierarchical Softmax – visual example

- Ideal of the tree is balance tree (explanation on white board)
  - Weighted matrix V shared through all training (and will be learned) connect hidden layer and all nodes in the tree
  - When generate tree, each word should have a binary number to point to the path from root to its position, hence, in each step, only compute the probability of node on its path

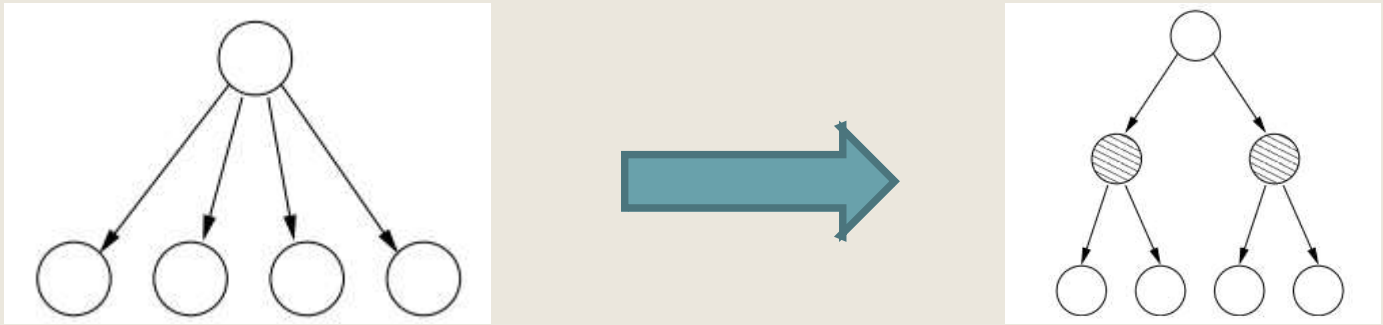




Hugo Lachorelle's explanation on Hierarchical Softmax
https://www.youtube.com/watch?v=B95LTf2rVWM

Hierarchical Softmax – building the tree

- Easiest way: randomly generate the tree to a balanced tree: decent result
- Use existing linguistic resources then re-construct the tree to as close to balanced tree as possible: WordNet (Bengio et al. 2005): using K-means



- Learn the hierarchy using a recursive portioning strategy:
  - Step 1: learn the representation once (using the random tree)
  - Step 2: using the word representation from step 1 for clustering algorithm to recursively divide the whole vocabulary to achieve balanced tree
  - This tree will have the closed words into same or closed nodes

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Warming up 1: draw sample from distribution

- **Prerequisite**: have pseudo random to take a random number from uniform distribution of [0,1]: z
- **Objective**: draw a sample y which have distribution function p(y)
- **Method**: find a function f(.) that y = f(z) to transform z to y
- With y = f(z), probability of z in [z, z + dz] equals to prob of y in [y, y + dy], hence:

$$p(y)dy = p(z)dz \Rightarrow p(y) = p(z)\frac{dz}{dy} = \frac{dz}{dy} \; (p(z) = 1: uniform)$$

$$z = h(y) \equiv \int_{-\infty}^{y} p(y')dy'$$

$$y = h^{-1}(z)$$

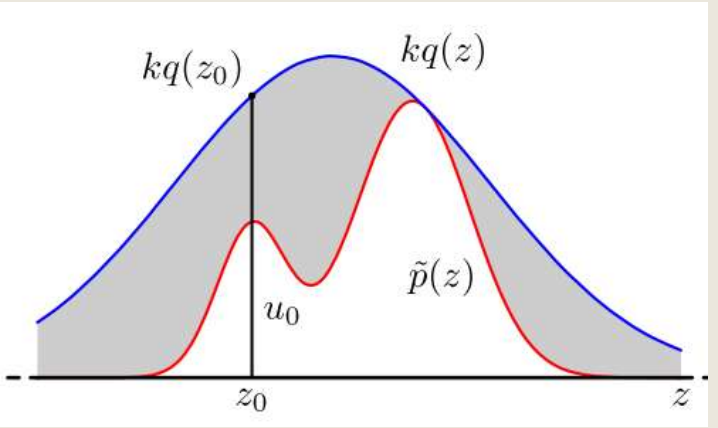
- **Steps**:
  - Step 1: compute $h(y) \equiv \int_{-\infty}^{y} p(y')dy'$
  - Step 2: resolve back $y = h^{-1}(z)$
  - Step 3: take random z, compute y

Warming up 2: Monte Carlo rejection sampling

- **Objective**: compute the distribution $p(z)$ which is not simple or more of the case, is the normalized distribution of known $\widetilde{p(z)}$:

$$p(z) = \frac{1}{Z_P}\widetilde{p(z)}$$

- Approximate $p(z)$ to a more simple normalized $q(z)$
- With constant k: $kq(z) \geq p(z) \forall z$
- Draw z_o from $q(z)$ and u_0 from [0,1] uniform
- If $u_0 > \widetilde{p(z_0)}$ then sample rejected, otherwise accepted
- Each pair of accepted (z_o, u_0) has uniform distribution under the curve of $\widetilde{p(z)}$



Pattern recognition and machine learning (Bishop)

- Repeat for a number of time to get the prob of accepted points, multiple that prob to $kq(z)$ to get the approximated normalized $p(z)$

Importance Sampling: method

- Objective: approximate expectation of function f(z) where z follow distribution p(z):

$$\mathbb{E}[f] = \sum_{l=1}^{L} p(z^l) f(z^l)$$

- Method: use proposal simple q(z), sample z^l from q(z):

$$\mathbb{E}[f] \cong \sum_{l=1}^{L} \frac{p(z^l)}{q(z^l)} f(z^l) : r_l = \frac{p(z^l)}{q(z^l)} \text{ is importance weights}$$

- Normally, normalize p(z), q(z) is expensive, use the unnormalized form $\widetilde{p(z)}$, $\widetilde{q(z)}$:

$$\mathbb{E}[f] = \sum_{l=1}^{L} w_l f(z^l)$$

$$w_l = \frac{\widetilde{p(z^l)}/\widetilde{q(z^l)}}{\sum \widetilde{p(z^m)}/\widetilde{q(z^m)}}$$

Importance Sampling: in word embedding

- Objective: compute the probability of word w given context c

$$P(w|C) = \frac{e^{-\varepsilon(w,C)}}{\sum_{v \in V} e^{-\varepsilon(v,C)}}, \varepsilon(w,C): compute\ from\ neural\ network$$

- Objective function: $J(\theta) = \sum \log\big(P(w|C)\big) = -\varepsilon(w,C) - \log(\sum e^{-\varepsilon(v,C)})$
- More importance, gradient of objective function:

$$\nabla J(\theta) = -\nabla\varepsilon(w,C) + \sum \frac{e^{-\varepsilon(v,C)}}{\sum_{v' \in V} e^{-\varepsilon(v',C)}} \nabla\varepsilon(v,C) = -\nabla\varepsilon(w,C) + \sum P(v|C)\, \nabla\varepsilon(v,C)$$

- Use importance sampling to compute expensive: $\sum P(v|C)\, \nabla\varepsilon(v,C)$

Algorithm 3 Importance Sampling Gradient Approximation

Add positive contribution: $\frac{\partial \mathcal{E}(w_t, h_t)}{\partial \theta}$
vector $a \leftarrow \mathbf{0}$
$b \leftarrow 0$
repeat N **times**
  Sample $w' \sim Q(\cdot|h_t)$
  $r \leftarrow \frac{e^{-\mathcal{E}(w', h_t)}}{Q(w'|h_t)}$
  $a \leftarrow a + r \frac{\partial \mathcal{E}(w', h_t)}{\partial \theta}$
  $b \leftarrow b + r$
Add negative contribution: $-\frac{a}{b}$

Bengio et al. 2003

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Noise Contrastive Estimation (NCE)

- **Idea**: simplify softmax to sigmoid
  - For multi-classes classification problem, softmax represents a way to normalized the probability of one output to be one class among all classes
  - Instead, if we consider target word as class True, all other words as class False then become linear regression with sigmoid, which is way cheaper in computing
- **Practice**: with each target word w_t on context h, generate k noise sample from Vocabulary which follow distribution $P_n(v)$, train the network to discriminate the True data with the noise
- Math: prob of true data given context is $P_d^h(w)$, prob of noise $P_n(w)$ (can be bigram, uniform but in paper, they chose unigram), approximated data prob with parameter $P_\theta^h(w)$.
  Objective function: $J^h = E_{P_d^h}\left[log \frac{P_\theta^h(w)}{P_\theta^h(w)+kP_n(w)}\right] + E_{P_n}\left[log \frac{kP_n(w)}{P_\theta^h(w)+kP_n(w)}\right]$
- Gradient:

$$\nabla J^{h,w}(\theta) = \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)} \nabla \log P_\theta^h(w) \; - \sum_{i=1}^{k} \frac{P_\theta^h(x_i)}{P_\theta^h(w) + kP_n(x_i)} \nabla \log P_\theta^h(x_i)$$

- Global objective: $J(\theta) = \sum_h P(h) J^h(\theta)$

NCE: implement or hyper-parameter

- Normalization term: $P(w|C) = \frac{e^{-\varepsilon(w,C)}}{\sum_{v \in V} e^{-\varepsilon(v,C)}}$, $\sum_{v \in V} e^{-\varepsilon(v,C)}$ can be considered as another parameter Z to learn. It is shown that set Z=1 does not effect the performance of learning => $P(w|C) = e^{-\varepsilon(w,C)}$, P(h) =1
- Number of noise sample: 25 is sufficient to achieve performance of softmax, hence speed-up factor about 45 for Vocabulary size of 10K, embedding dimension of 100
- Noise distribution: unigram out perform uniform
- Compare to importance sampling:
  - NCE weights: $\frac{P_\theta^h(x_i)}{P_\theta^h(w)+kP_n(x_i)}$ always between [0,1]
  - Importance sampling: $w_l = \frac{\widetilde{p(z^l)}/\widetilde{q(z^l)}}{\sum \widetilde{p(z^m)}/\widetilde{q(z^m)}}$ can have some dominant element with specific p(z) and q(z) hence training is not stable

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Negative sampling: approximation of NCE

- Idea: simplify the most expensive calculate of $kP_n(x_i)$ in NCE by setting it to 1 (so that the prob form will be exactly sigmoid function)
- Method: generate k noise sample, prob of data is positive:

$$P(y=1|w,C) = \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)} = \frac{P_\theta^h(w)}{P_\theta^h(w) + 1} = \sigma(\varepsilon(w,C))$$

- Objective function:

$$J_\theta = -\sum_{w_i \in V} [log\, \sigma(\varepsilon(w,C)) + \sum_i^k \mathbb{E}_{w_i \sim P_n(w)} \log(\, \sigma(-\varepsilon(w_i,C)))$$

- With w_i drawn from distribution P_n, using Monte Carlo approximation we have:

$$J_\theta = -\sum_{w_i \in V} [log\, \sigma(\varepsilon(w,C)) + \sum_i^k \log(\, \sigma(-\varepsilon(w_i,C)))$$

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Word2Vec: model

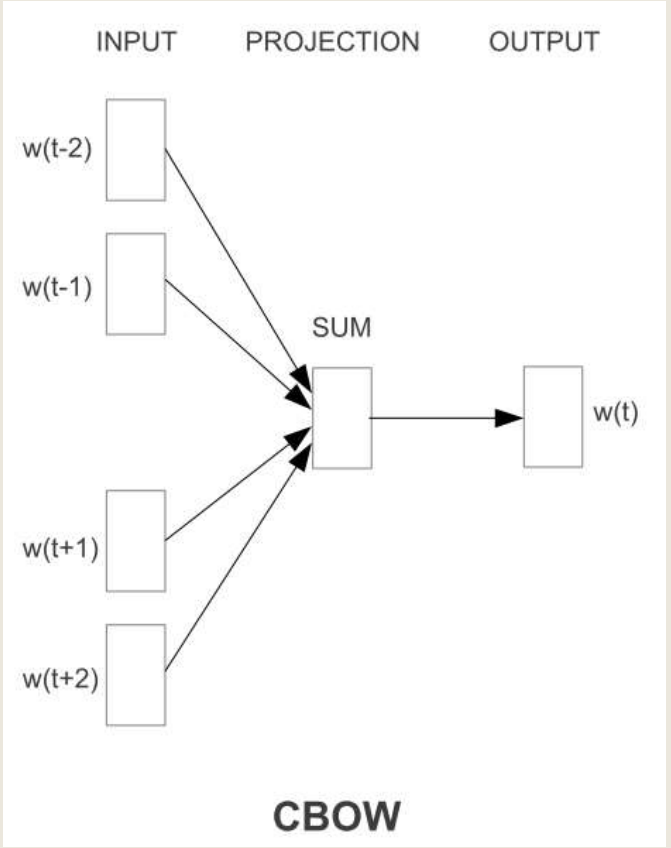
- Simple model: Input -> Shared weights embedding layer -> output (no non-linear activation function)
- Context: not only previous words of target word in a sentence but also the words after that as a window: w_{t-n}, …,w_t, …, w_{t+n}
- Using Negative sampling to compute output probability to speed up training with the main purpose is to gain better word representation, not other tasks
- Contains 2 main models: Continuous Bag of Words (CBOW) and Continuous Ski-gram

Word2Vec: Continuous Bag of Words

- Each time step t, take n words before and n words behind
- Input 2n words go through projection layer (embedding layer) then Averaged to get a single input vector
- Objective function:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \log(\, p(w_t | w_{t-n}, \dots, w_{t+n}))$$

- No hidden layer, prob of output computed directly by inner product of output vector and input vector



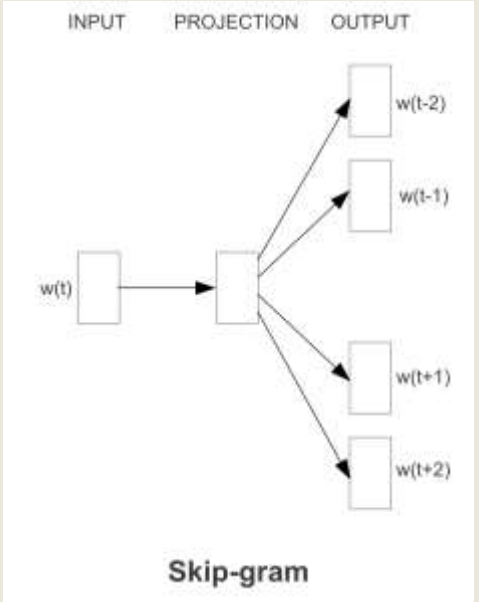
Word2Vec: Skip-gram

- With each target word, predict the surrounding 2n words
- Objective:

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

- Target word is input w_I, predicted is output w_O:

$$p(w_O|w_I) = \frac{e^{(v'^{T}_{wO} v_{wI})}}{\sum e^{(v'^{T}_{w} v_{wI})}}$$



Word2Vec: other tricks

- Preprocessing:
  - Dynamic window size: with max size of C, will be 2R words sample random from window size 2C
  - Add more weights for closer word to target words since they have more relation
  - Supsampling frequent words: intentionally remove high frequent words from corpus before generating context for words
  - Deleting rare words before generating context for words
- Softmax approximation:
  - Simplified NCE to Negative sampling
- Generate noise samples:
  - Compute frequency of words $f(w_i)$ then use noise distribution $P(w) = \frac{f(w)^{3/4}}{\sum f(w)^{3/4}}$
  - Create a huge index table, put index of words with number of duplicate proportional to distribution P(w)
  - Randomly generate index number then take the word from index table

Agenda

1. Brief Intro on Word Embedding
2. Several algorithms to approximate softmax
   - Hierarchical Softmax
   - Importance Sampling
   - Noise Contrastive Estimation (NCE)
   - Negative Sampling
3. State of the art models
   - Word2Vec
   - Fasttext

Fasttext: word embedding version (2017)

- Extension of Word2Vec
- Idea: use n-gram to create subwords then represent target words by sum of all of its subwords representation
- General model:
  - Skip-gram, context window size between 1 and 5
  - Sub-sample frequent word with rejection threshold of 1e-4
  - Negative sampling with number of noise sample is 5, use square root of unigram frequency as noise distribution

Fasttext: subword model

- Skip-gram objective function

$$J = \frac{1}{T}\sum_{t=1}^{T}[\sum_{c\in Ct} l\big(s(w_t, w_c)\big) + \sum_{n\in N} l(-s(w_t, n))]\,, s(u, v) = u^T v$$

- Subword:
  - Each word is added with <, > as start and end of word
  - Generate n-gram for that word and include that word it self as sub-set

    Example “where”, 3-gram: <where> -> <wh, whe, her, ere, re>, <where>
  - Each n-gram will be represented with a vector as a set of new dictionary G
  - Vector for w is sum of all vector represent g generated by w
  - Scoring function: $s(w, c) = \sum_{g\in G_w} z_g^T v_c$

References

1. http://ruder.io/word-embeddings-1/ (as well as part 2, 3 of the blog sequence)
2. Bengio et al. A Neural Probabilistic Language Model
3. Mikolov et al. Efficient Estimation of Word Representation in Vector Space
4. Mikolov et al. Distributed Representations of Words and Phrases and their Compositionality
5. Pennington et al. GloVe: Global Vectors for Word Representation
6. Christopher M. Bishop. Pattern Recognition and Machine Learning
7. Dan Jurafsky: Language Modeling (Stanford material)
8. P. Bojanowski et al. Enriching Word Vectors with Subword Information
9. A. Joulin et al. Bag of Tricks for Efficient Text Classification
10. A. Joulin et al. FastText.zip: Compressing text classification models
11. Morin, F., & Bengio, Y. (2005). Hierarchical Probabilistic Neural Network Language Model
12. http://adventuresinmachinelearning.com (tutorials of w2v on tensorflow and keras)
13. https://github.com/tensorflow/tensorflow/blob/r1.8/tensorflow/examples/tutorials/word2vec
14. https://github.com/facebookresearch/fastText
15. https://www.youtube.com/watch?v=B95LTf2rVWM