Word Embedding 深入理解

Well-known models for learning word embeddings based on language modelling. 这些无监督方法学得的词向量已经在很多NLP任务中代替了传统的“分布式特征”（distributional feature，since the underlying semantic theory is called distributional semantics）,比如Brown clusters以及LSA 特征。如今盛行的Word Embedding都是指的“dense representations of words in a low-dimensional vector space”，还可以表示为*word vectors* , *distributed representations*, semantic vector space or simply word space。Word Embedding表示的向量能表示语义相似性，可以用来表示自身的语义特征，计算不同词之间的语义相似性，也可以作为一种表示，应用到其他NLP任务中去，like text classification, document clustering, part of speech tagging, named entity recognition, sentiment analysis, and so on.

Distributional Semantics History

Idea: 相对于形式化的语言学，语言上下文信息能形成一个合理的关于语言项（如word）的表示。

19世纪八九十年代就有这种semantic representation的应用。Methods for using **automatically generated contextual features** were developed more or less simultaneously around 1990 in several different research areas. LSA/LSI就是最有影响力的模型，也是现今topic models的先驱；与此同时也有利用这些contextual representation的人工神经网络模型，如SOM,SRN，他们是如今**神经语言模型**的先驱。

后来的模型几乎都是对原有模型的一些细微改良，比如topic models里面的PLSA,LDA都是对LSA的改良；neural language moddel基于SRN、CNN、Autoencoders等。Distrubutional semantic models都是基于HAL等。主要区别是他们使用的是不同类型的contextual information，而不同的上下文信息表达了不同的语义相似性。

> *The main difference between these various models is the type of contextual information they use. LSA and topic models use* documents *as contexts, which is a legacy from their roots in information retrieval. Neural language models and distributional semantic models instead use* words *as contexts, which is arguably more natural from a linguistic and cognitive perspective. These different contextual representations capture different types of semantic similarity; the document-based models capture* ***semantic relatedness*** *(e.g. “boat” – “water”) while the word-based models capture* ***semantic similarity*** *(e.g. “boat” – “ship”). This very basic difference is too often misunderstood.*

这里对于普遍的错误认识还有两个点需要说明：

- 没必要为了得到好的word embedding而使用**深度**神经网络模型，因为目前的CBOW和Skip-gram模型都是和SRN相似的浅层神经网络模型，却已经取得了比较成功的效果。
- There is no qualitative difference between (current) **predictive* neural network models** and ***count-based*** **distributional semantics models.** Rather, they are different computational means to arrive at the same type of semantic model。

那么现在最新的一些语义模型怎么样呢？

What about the current state of the art? The boring answer is that it depends on what task you want to solve, and how much effort you want to spend on optimizing the model.（现在的模型取决于你要解决什么样的任务以及你对模型的优化做了多大努力） The somewhat more enjoyable answer is that you will probably do fine whichever current method you choose, since they are more or less equivalent（无论你选择哪个模型，你都能取得差不多较好的效果，因为他们多多少少都是等同的）. A good bet is to use a **factorized model** – either using explicit factorization of a distributional semantic model (available in e.g. the PyDSM python library or the GloVeimplementation), or using a neural network model like those implemented in word2vec – since they produce state of the art results (Österlund et al., 2015) and are robust across a wide range of semantic tasks (Schnabel et al., 2015).

Word Embedding History

2003, Bengio et al.提出术语“word embedding”,并训练神经网络参数；

2008年，Collobert and Weston 提出预训练word embedding的应用；

2013年，Mikolov et al. 提出了最受欢迎的 **word2vec** 允许无缝训练以及使用预训练word embedding;

2014年，Pennington et al. 放出GloVe.

词向量是少数无监督学习的成功应用。 they don't require expensive annotation, can be derived from large unannotated corpora that are readily available. Pre-trained embeddings can then be used in downstream tasks that use small amounts of labeled data.

Word embedding models

Embedding Layer

在神经网络中，语料通过词典表示后作为输入，然后通过网络第一层将输入embed成低维向量，这个第一维就是Embedding Layer。但这些方法和Word2vec是有所区别的，前者将word embedding当作一个副产品输出，而后者则明确的将Word embedding的生成作为目标，两者的区别就是：

1）计算复杂度的不一样，计算复杂度是一个非常关键的问题，通过深度神经网络来生成word embedding是很昂贵的，这也就是为什么2013年才火起来的原因。

2）训练目标不一样。word2vec和glove是产生的通用word embedding relationship，对那些不依赖这种相似性的任务来说，并不能产生太大帮助的，比如boat和ship比较具有相关性，而有些任务中可能需要boat和water具有相关性；但是regular neural network不一样，他们产生的是task-specific的embeddings。对于依赖于semantically coherent representations的任务，如语言模型，会和word embedding models产生相似的word embedding。

Word embedding models are often evaluated using perplexity, a cross-entropy based measure borrowed from language modelling.

notational standards

- T training words $w_1, w_2, w_3, \cdots, w_T$
- vocabulary V
- context of n words
- input embedding v_w
- input embedding v'_w
- model outputs some score $f_\theta(x)$
- Objective function J_θ

Language Modelling

Task:基于n-gram语言模型： 给定n-1个前面的词，计算下一个词的概率：$p(w_t \mid w_{t-1}, \cdots w_{t-n+1})$

基于链式法则和Markov假设，整个sentence或者document的概率就是：

$$p(w_1, \cdots, w_T) = \prod_i p(w_i \mid w_{i-1}, \cdots, w_{i-n+1})$$

如何计算每个词的概率呢？可以根据n-gram出现的频率来计算：

$$p(w_t \mid w_{t-1}, \cdots, w_{t-n+1}) = \frac{count(w_{t-n+1}, \cdots, w_{t-1}, w_t)}{count(w_{t-n+1}, \cdots, w_{t-1})}$$

而在神经网络模型中，就是通过下面softmax计算得到：

$$p(w_t \mid w_{t-1}, \cdots, w_{t-n+1}) = \frac{\exp(h^\top v'_{w_t})}{\sum_{w_i \in V} \exp(h^\top v'_{w_i})}$$

$h^\top v'_{w_t}$ computes the (**unnormalized**) **log-probability** of word w_t, h是网络倒数第二层的输出, we normalize by the sum of the log-probabilities of all words in V,which is computation expensive.

目标函数:

$$
\begin{aligned}
J_\theta &= \frac{1}{T}\log p(w_1, \cdots, w_T) \\
&= \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})
\end{aligned}
$$

Classic neural language model

Their model maximizes what we've described above as the prototypical neural language model objective (we omit the regularization term for simplicity):

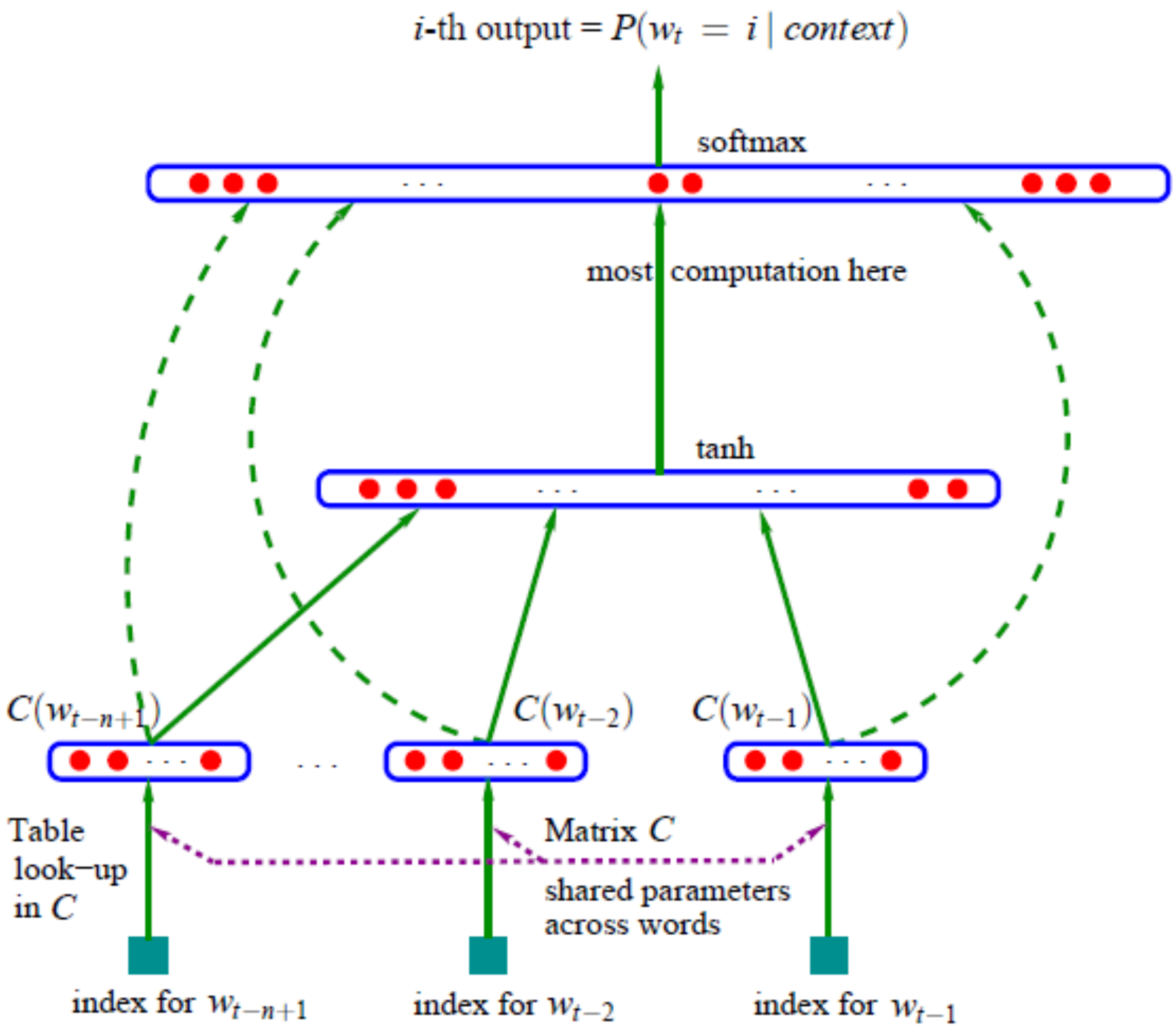


Figure 2: Classic neural language model (Bengio et al., 2003)

The general building blocks of their model, however, are still found in all current neural language and word embedding models. These are:（这些层现在的模型中还被使用）

1. **Embedding Layer**: a layer that generates word embeddings by multiplying an index vector with a word embedding matrix;
2. **Intermediate Layer(s)**: one or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of nn previous words;// 这一项可以用LSTM代替
3. **Softmax Layer**: the final layer that produces a probability distribution over words in V.//这一项的计算量非常大，one of the key challenges both in neural language models as well as in word embedding models.

C&W model

Word embeddings trained on **a sufficiently large dataset** carry syntactic and semantic meaning and improve performance on downstream tasks.为了解决计算量的问题，他们使用了新的loss，ranking objective，而不再使用softmax+CE，这个loss能为正确的单词序列产生比错误序列更高的分数，他们使用了一个pairwise ranking criterion：

$$J_\theta \ = \sum_{x\in X}\sum_{w\in V}\max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\}$$

这里是一个hinge loss，其中x^w是采样的负样本，x为正样本。正样本分数要比负样本的分数至少高1.

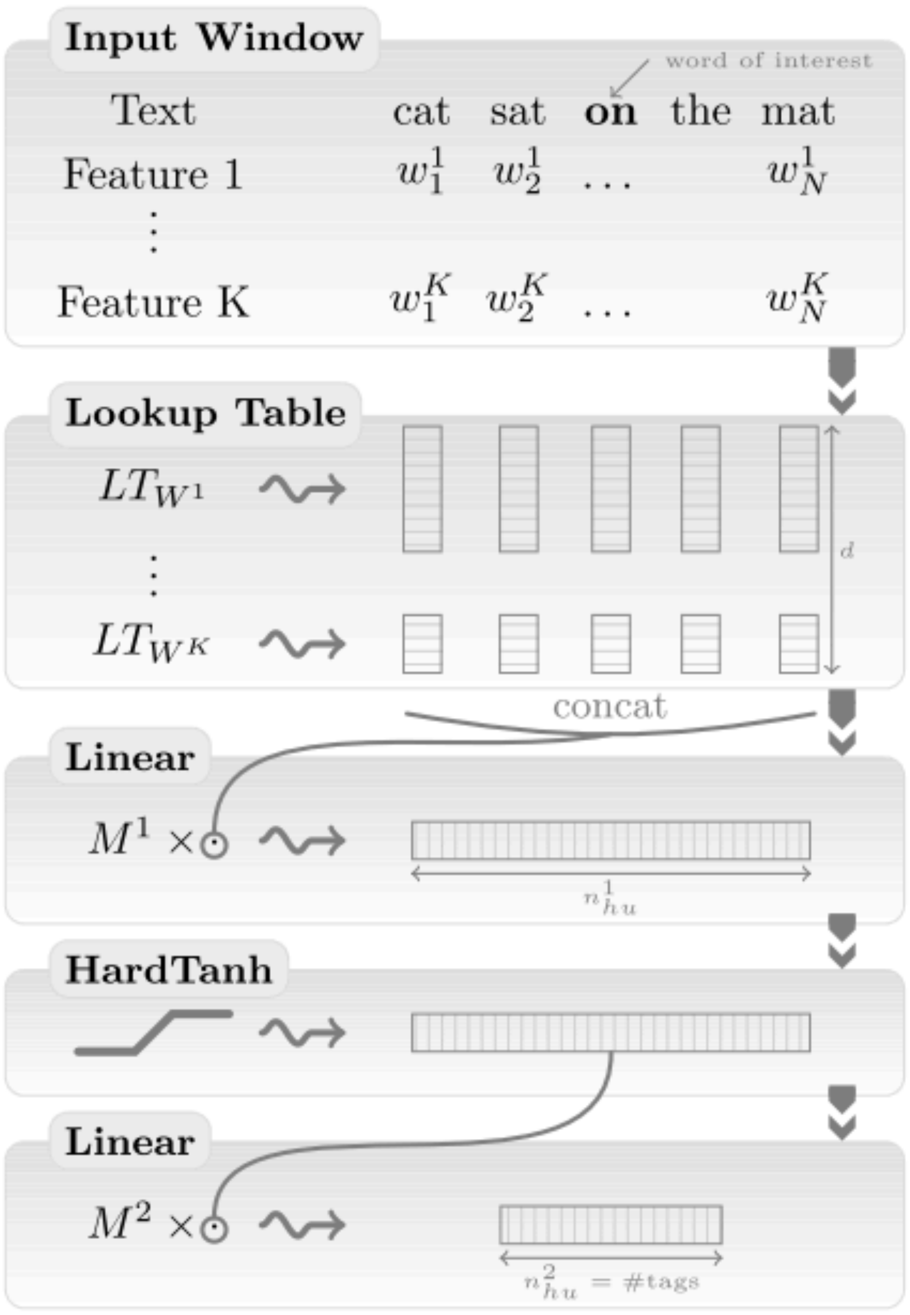


Figure 3: The C&W model without ranking objective (Collobert et al., 2011)

Word2Vec

word2vec is not be considered to be part of deep learning, as its architecture is neither deep nor uses non-linearities (in contrast to Bengio's model and the C&W model).

In their first paper [7], Mikolov et al. propose two architectures for learning word embeddings that are computationally less expensive than previous models. In their second paper [8], they improve upon these models by employing additional strategies to enhance training speed and accuracy.

模型的两个优点：

- 没有计算昂贵的隐藏层
- 能够考虑额外的上下文信息

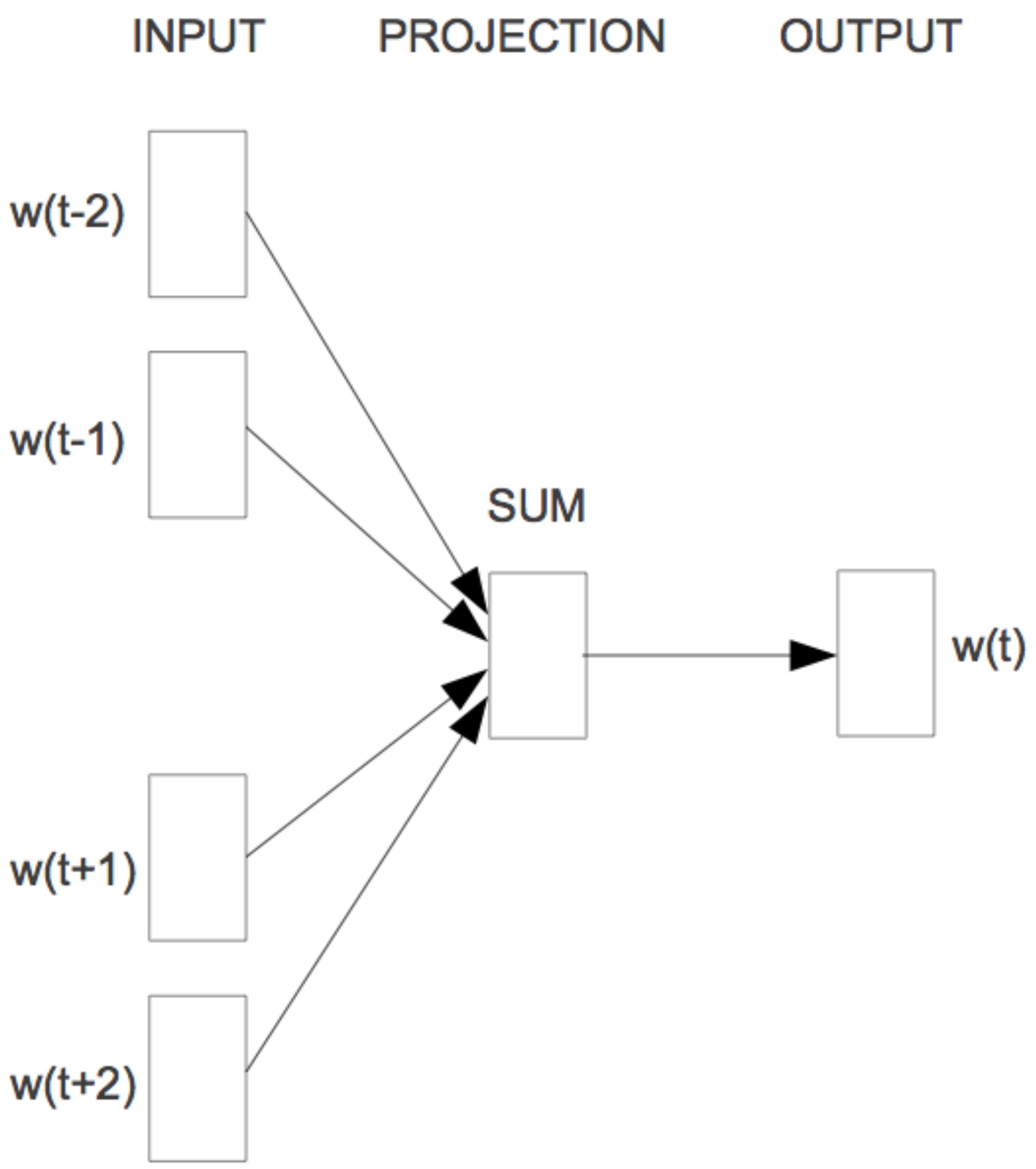
结果的成功主要是在于训练技巧上的策略。

CBOW

生成word embedding不受模型所能看到的词的限制：

> *In their first paper [7], Mikolov et al. propose two architectures for learning word embeddings that are computationally less expensive than previous models. In their second paper [8], they improve upon these models by employing additional strategies to enhance training speed and accuracy.*

因此这个模型中考虑了每个词前后各n个词：

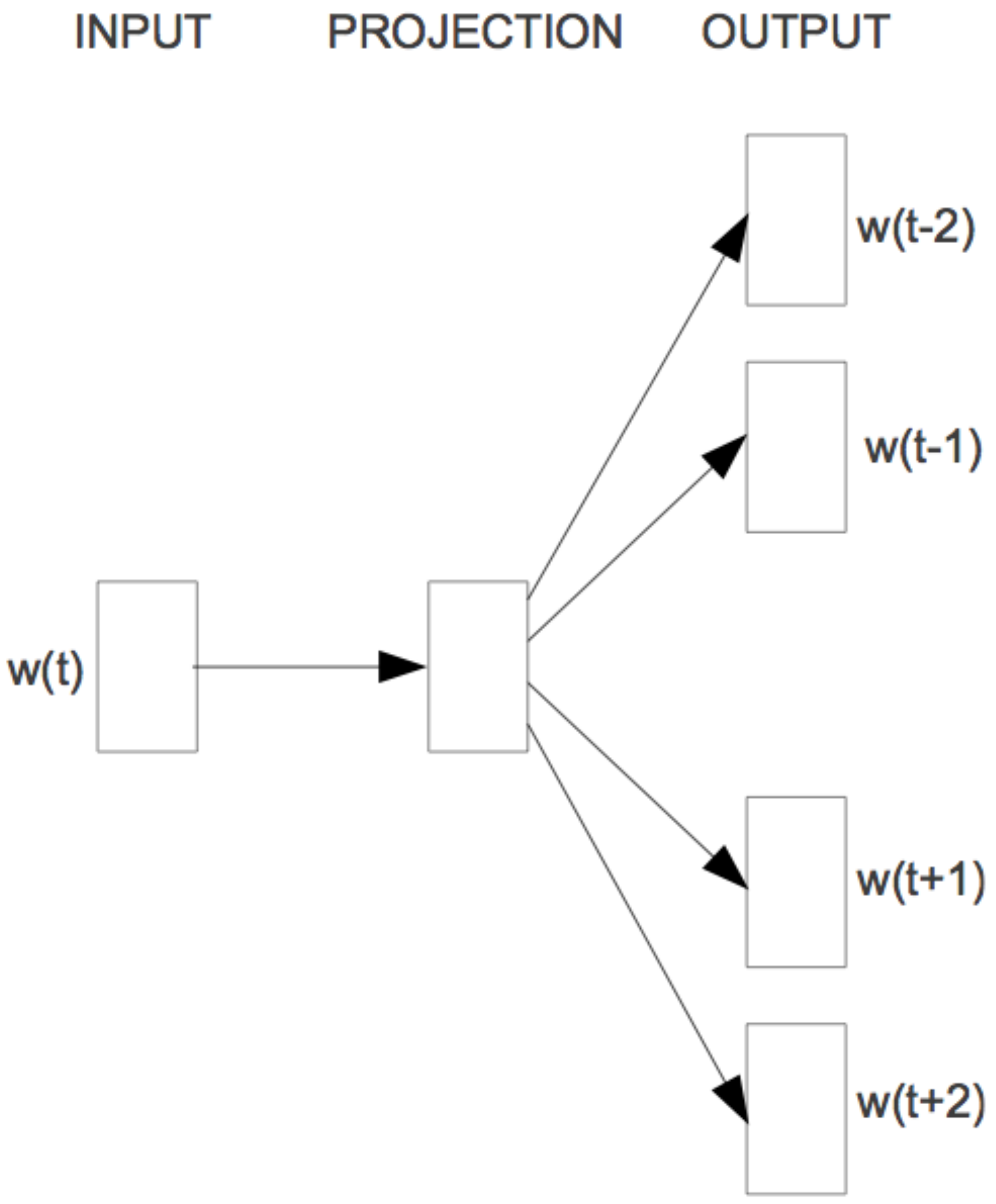


目标函数也有变化,多考虑了n个词:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t \mid w_{t-n}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+n})$$

Skip-gram

kip-gram turns the language model objective on its head: Instead of using the surrounding words to predict the centre word as with CBOW, skip-gram uses the centre word to predict the surrounding words.



目标方程的变化：The skip-gram objective thus sums the log probabilities of the surrounding nn words to the left and to the right of the target word w_t to produce the following objective:

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

$p(w_{t+j} \mid w_t)$的计算：

$$p(w_{t+j} \mid w_t) = \frac{\exp(h^\top v'_{w_{t+j}})}{\sum_{w_i \in V} \exp(h^\top v'_{w_i})}$$

注意，这个地方w_t只是获取embedding的输入，具体计算$p(w_{t+j} \mid w_t)$只和正样本w_{t+j}及整个词典数据有关，前面$p(w_t \mid w_{t-1}, \cdots, w_{t-n+1})$也是一样。

skip-gram architecture不包含隐藏层，不存在h, 因此h is simply the word embedding v_{w_t} of the input word w_t. This also makes it clearer why we want to have different representations for input embeddings v_w and output embeddings v'_w, as we would otherwise multiply the word embedding by itself. Replacing h with v_{w_t} . Note that the notation in Mikolov's paper differs slightly from ours, as they denote the centre word with w_I and the surrounding words with w_O. If we replace w_t with w_I, $w_{t+j}$with w_O, and swap the vectors in the inner product due to its commutativity, we arrive at the softmax notation in their paper:

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

Referrence

https://www.gavagai.se/blog/2015/09/30/a-brief-history-of-word-embeddings/

http://ruder.io/word-embeddings-1/index.html#skipgram