Learning word embeddings efficiently with noise-contrastive estimation

Andriy Mnih & Koray Kavukcuoglu
DeepMind Technologies

Overview

- Learning flexible word representations is the first step towards learning semantics.
- The best current approach to learning word embeddings involves training a neural language model to predict each word in a sentence from its neighbours.
  - Need to use a lot of data and high-dimensional embeddings to achieve competitive performance.
  - More scalable methods translate to better results.
- We propose **a simple and scalable approach to learning word embeddings** based on training lightweight models with noise-contrastive estimation.
  - It is **simpler, faster, and produces better results than the current state-of-the art method**.

Neural probabilistic language models

- NPLMs quantify the compatibility between a context h and a target word w using a scoring function $s_\theta(w, h)$.
- The distribution of the target word is defined in terms of scores: $P_\theta^h(w) = \exp(s_\theta(w, h))/Z_\theta(h)$, where $Z_\theta(h) = \sum_{w'} \exp(s_\theta(w', h))$ is the normalizer for context h.
- The scoring function is parameterized in terms of word embeddings which are treated as model parameters.
- Training an NPLM is usually slow because
  1. The complexity of the ML parameter update is linear in the vocabulary size (due to $Z_\theta(h)$).
  2. $s_\theta(w, h)$ is expensive to evaluate.

Learning word embeddings efficiently

We learn word embeddings efficiently by

1. Training the models using noise-contrastive estimation instead of maximum likelihood.
2. Using simplified log-bilinear models that compute $s_\theta(w, h)$ efficiently using only vector-vector operations.

Vector log-bilinear model (vLBL)

- Predict the word in the middle of an $n+1$-word window from the n-word context $h = w_1, ..., w_n$ surrounding it.
- Each word has two embeddings: a conditional embedding r_w and a target embedding q_w.
- The score is computed based on the average of the context word representations:

$$s_\theta(w, h) = \left(\frac{1}{n}\sum_{i=1}^{n} r_{w_i}\right)^\top q_w + b_w.$$

Inverse vector log-bilinear model (ivLBL)

- Predict n words in an $(n+1)$-word window from its middle word (independently).
- The score for predicting the i^{th} word in the window from the middle word w is:

$$s_\theta^i(w_i, w) = r_w^\top q_{w_i} + b_{w_i}.$$

- Both vLBL and ivLBL can be extended to take into account the relative positions of words in the window.

Noise-contrastive estimation

- Idea: Fit a density model by learning to discriminate between samples from the data distribution and samples from a known noise distribution (Gutmann and Hyvärinen, 2010).
- Training language models with **NCE makes training time effectively independent of the vocabulary size** (Mnih and Teh, 2012).
- Assuming noise samples are k times more frequent than data samples, the posterior probability of a sample (in context h) being from the data distribution is

$$P^h(D = 1|w) = \frac{P_d^h(w)}{P_d^h(w) + kP_n(w)}.$$

- Using $\exp(s_\theta(w, h))$ in place of $P_d^h(w)$, we model this as

$$P^h(D = 1|w, \theta) = \sigma\left(s_\theta(w, h) - \log kP_n(w)\right),$$

where $\sigma(x)$ is the logistic function and $\Delta s_\theta(w, h) = s_\theta(w, h) - \log(kP_n(w))$ is the difference in the scores of word w under the (unnormalized) model and the (scaled) noise distribution.

- We fit the model by maximizing the $\log$-posterior probability of the correct labels D, averaged over the data and noise distributions:

$$\begin{aligned} J^h(\theta) =& E_{P_d^h}\left[\log P^h(D = 1|w, \theta)\right] + \\ & kE_{P_n}\left[\log P^h(D = 0|w, \theta)\right], \end{aligned}$$

- The contribution of a word / context pair w, h to the gradient of the objective is estimated using sampling:

$$\begin{aligned} \frac{\partial}{\partial\theta} J^{h,w}(\theta) =& (1 - \sigma\left(\Delta s_\theta(w, h)\right)) \frac{\partial}{\partial\theta} \log P_\theta^h(w) - \\ & \sum_{i=1}^{k}\left[\sigma\left(\Delta s_\theta(x_i, h)\right) \frac{\partial}{\partial\theta} \log P_\theta^h(x_i)\right]. \end{aligned}$$

where $\{x_i\}$ are k samples from the noise distribution, typically a unigram estimated from the training data.

MSR Sentence Completion Challenge

Task: given a sentence with a missing word, pick the correct completion from a list of candidate words.

- Training set: 522 19th-century novels (47M words)
- Test set: 1,040 sentences from five Sherlock Holmes novels
- Five candidate completions per sentence.

Model	Context	Latent dim.	Percent correct
Skip-gram	10L+10R	640	48.0
LSA	sentence	300	49
LBL	10L	300	54.7
ivLBL	5L+5R	100	51.0
ivLBL	5L+5R	300	55.2
ivLBL	5L+5R	600	**55.5**

Word analogy-based evaluation

- Two analogy-based word similarity tasks recently released by Google and Microsoft Research.
- Questions of the form "a is to b is as c is to _".
  - For example, $London : England \rightarrow Kiev : \ ?$.
  - The task is to identify the held-out fourth word.
  - Only exact word matches count (synonyms not accepted).
- We answer $a : b \rightarrow c : \ ?$ by finding word d^* with the representation most similar to $\vec{b} - \vec{a} + \vec{c}$ according to cosine similarity:

$$d^* = \arg\max_{\vec{x}} \frac{(\vec{b} - \vec{a} + \vec{c})^\top \vec{x}}{\|\vec{b} - \vec{a} + \vec{c}\|},$$

where all representation vectors are normalized.

- Evaluation metric: accuracy (percent correct).

NCE vs. tree-based training

- Training data: Wikipedia from April 2013
  - 1.5B words, 870K-word vocabulary
- Experimental setup (unless stated otherwise):
  - Models: 100D word embeddings, predict 5 words on both sides, no position-dependent weights.
  - NCEk denotes NCE training using k noise samples.

Model	Google Semantic	Google Syntactic	Google Overall	MSR	Time (hours)
Skip-gram(*)	28.0	36.4	32.6	31.7	12.3
ivLBL+NCE1	28.4	42.1	35.9	34.9	3.1
ivLBL+NCE2	30.8	44.1	38.0	36.2	4.0
ivLBL+NCE3	34.2	43.6	39.4	36.3	5.1
ivLBL+NCE5	37.2	44.7	41.3	**36.7**	7.3
ivLBL+NCE10	38.9	45.0	42.2	36.0	12.2
ivLBL+NCE25	**40.0**	**46.1**	**43.3**	**36.7**	26.8

Large-scale evaluation

- Comparison of vLBL and ivLBL models to their hierarchical counterparts from (Mikolov et al., 2013).
- ivLBL models: Predict 5 words on each side of the current word.
  - Trained with NCE25 using AdaGrad.
- vLBL models: Predict the current word from the 5 words before and 5 after it.
  - Trained with NCE5 using AdaGrad.
- No position-dependent weights were used.

Model	Embed. Dim.	Train. words	Google Sem.	Google Syn.	Google Overall	MSR	Time (days)
Skip-gram	300	1.6B	52.2	55.1	53.8		2.0
Skip-gram	300	785M	56.7	52.2	55.5		2.5
Skip-gram	1000	6B	**66.1**	65.1	65.6		2.5×125
ivLBL	100	1.5B	52.6	48.5	50.3	39.2	1.2
ivLBL	100	1.5B	55.9	50.1	53.2	42.3	2.9
ivLBL	100×2	1.5B	59.3	54.2	56.5	44.6	2.9
ivLBL	300	1.5B	61.2	58.4	59.7	48.8	1.2
ivLBL	300	1.5B	63.6	61.8	62.6	52.4	4.1
ivLBL	300×2	1.5B	65.2	63.0	64.0	54.2	4.1
CBOW	300	1.6B	16.1	52.6	36.1		0.6
CBOW	1000	6B	57.3	68.9	63.7		2×140
vLBL	300	1.5B	40.3	55.4	48.5	48.7	0.3
vLBL	100	1.5B	45.0	56.8	51.5	52.3	2.0
vLBL	300	1.5B	54.2	64.8	60.0	58.1	2.0
vLBL	600	1.5B	57.3	66.0	62.1	59.1	2.0
vLBL	600×2	1.5B	60.5	67.1	64.1	60.8	3.0
vLBL	600×2	1.5B	63.0	**69.8**	**66.7**	**62.8**	7.0

How do more expressive models perform?

- We can make our models more expressive by making them aware of word order through position-dependent weights. Does this lead to better word embeddings?
- Dataset: MSR Gutenberg corpus
  - 47M words, 80K-word vocabulary
- Training: 20 epochs of NCE5 with AdaGrad.
- (D)/(I) denotes models with/without position-dependent weights.
- Left/middle/right column gives the in-vocabulary accuracy obtained using the conditional/target/both word embeddings.
- nL/nR denotes n words on the left/right of the current word.

Model	Context	Google C	Google T	Google B	MSR C	MSR T	MSR B	Time (h)
vLBL(D)	5L + 5R	25.3	24.6	30.9	**29.9**	29.5	36.1	2.6
vLBL(D)	10L	22.4	16.2	32.2	26.7	11.5	**40.7**	2.6
vLBL(D)	10R	14.6	24.7	32.3	11.3	29.4	40.6	2.6
vLBL(I)	5L + 5R	**28.5**	**30.3**	31.3	29.3	**30.9**	32.7	2.3
vLBL(I)	10L	24.2	17.5	31.6	25.3	12.9	32.2	2.3
vLBL(I)	10R	17.2	25.3	**32.5**	12.8	25.9	34.0	2.1
ivLBL(D)	5L + 5R	16.5	14.1	17.7	18.5	17.9	21.3	1.2
ivLBL(I)	5L + 5R	27.6	27.4	31.0	27.3	26.8	31.4	1.2