A fast and simple algorithm for training neural probabilistic language models

Andriy Mnih

Joint work with Yee Whye Teh

Gatsby Computational Neuroscience Unit
University College London

25 January 2013

Statistical language modelling

- **Goal**: Model the joint distribution of words in a sentence.
- **Applications**:
  - speech recognition
  - machine translation
  - information retrieval
- **Markov assumption**:
  - The distribution of the next word depends on only a fixed number of words that immediately precede it.
  - Though false, makes the task much more tractable without making it trivial.

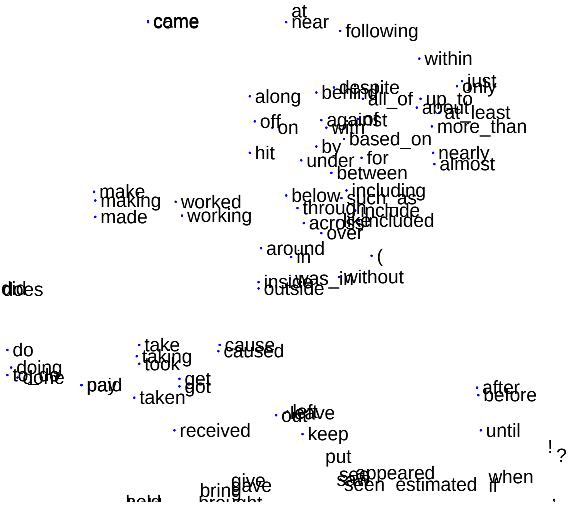
n-gram models

- **Task**: predict the **next word** w_n from $n-1$ preceding words $h = w_1, ..., w_{n-1}$, called the **context**.
- n-gram models are conditional probability tables for $P(w_n|h)$:
  - Estimated by counting the number of occurrences of each word n-tuple and normalizing.
  - Smoothing is essential for good performance.
- n-gram models are the most widely used statistical language models due to their simplicity and good performance.
- **Curse of dimensionality**:
  - The number of model parameters is exponential in the context size.
  - Cannot take advantage of large contexts.

Neural probabilistic language modelling

- Neural probabilistic language models (NPLMs) use **distributed representations** of words to deal with the curse of dimensionality.
- Neural language modelling:
  - Words are represented with real-valued **feature vectors learned from data**.
  - A neural network maps a context (a sequence of word feature vectors) to a distribution for the next word.
  - Word feature vectors and neural net parameters are learned jointly.
- NPLMs generalize well because smooth functions map nearby inputs to nearby outputs.
- Similar representations are learned for words with similar usage patterns.
- Main drawback: very long training times.

t-SNE embedding of learned word representations



Defining the next-word distribution

- A NPLM quantifies the **compatibility between a context h and a candidate next word w** using a scoring function $s_\theta(w, h)$.
- The **distribution for the next word** is defined in terms of scores:

$$P_\theta^h(w) = \frac{1}{Z_\theta(h)} \exp(s_\theta(w, h)),$$

  where $Z_\theta(h) = \sum_{w'} \exp(s_\theta(w', h))$ is the normalizer for context h.
- Example: **Log-bilinear model (LBL)** performs linear prediction in the space of word representations:
  - $\hat{r}(h)$ is the predicted representation for the next word obtained by linearly combining the representations of the context words:

$$\hat{r}(h) = \sum_{i=1}^{n-1} C_i r_{w_i}.$$

  - The scoring function is $s_\theta(w, h) = \hat{r}(h)^\top r_w$.

Maximum-likelihood learning

- For a single context, the gradient of the log-likelihood is

$$
\begin{aligned}
\frac{\partial}{\partial\theta}\log P_\theta^h(w) &= \frac{\partial}{\partial\theta}s_\theta(w,h) - \frac{\partial}{\partial\theta}\log Z_\theta(h) \\
&= \frac{\partial}{\partial\theta}s_\theta(w,h) - \sum_{w'} P_\theta^h(w')\frac{\partial}{\partial\theta}s_\theta(w',h).
\end{aligned}
$$

- Computing $\frac{\partial}{\partial\theta}\log Z_\theta(h)$ is expensive: the **time complexity is linear in the vocabulary size** (typically tens of thousands of words).
- Importance sampling approximation (Bengio and Senécal, 2003):
  - Sample words from a proposal distribution $Q^h(x)$ and reweight the gradients:

$$
\frac{\partial}{\partial\theta}\log Z_\theta(h) \approx \sum_{j=1}^{k}\frac{v(x_j)}{V}\frac{\partial}{\partial\theta}s_\theta(x_j,h)
$$

    where $v(x) = \frac{\exp(s_\theta(x,h))}{Q^h(x)}$ and $V = \sum_{j=1}^{k} v(x_j)$.
  - **Stability issues**: need either a lot of samples or an adaptive proposal distribution.

Noise-contrastive estimation

- **NCE 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).
- If noise samples are k times more frequent than data samples, the posterior probability that a sample came from the data distribution is

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

- To fit a model $P_\theta(x)$ to the data, use $P_\theta(x)$ in place of $P_d(x)$ and maximize

$$\begin{aligned} J(\theta) =& E_{P_d}\left[\log P(D=1|x,\theta)\right] + kE_{P_n}\left[\log P(D=0|x,\theta)\right] \\ =& E_{P_d}\left[\log \frac{P_\theta(x)}{P_\theta(x) + kP_n(x)}\right] + kE_{P_n}\left[\log \frac{kP_n(x)}{P_\theta(x) + kP_n(x)}\right]. \end{aligned}$$

The advantages of NCE

- **NCE allows working with unnormalized distributions** $P^u_\theta(x)$:
  - Set $P_\theta(x) = P^u_\theta(x)/Z$ and **learn** Z (or $\log Z$).
- The gradient of the objective is

$$\frac{\partial}{\partial\theta}J(\theta) = E_{P_d}\left[\frac{kP_n(x)}{P_\theta(x)+kP_n(x)}\frac{\partial}{\partial\theta}\log P_\theta(x)\right] - kE_{P_n}\left[\frac{P_\theta(x)}{P_\theta(x)+kP_n(x)}\frac{\partial}{\partial\theta}\log P_\theta(x)\right].$$

- **Much easier to estimate than the importance sampling gradient** because the weights on $\frac{\partial}{\partial\theta}\log P_\theta(x)$ are always between 0 and 1.
  - Can use far fewer noise samples as a result.

NCE properties

- The NCE gradient can be written as

$$\frac{\partial}{\partial\theta}J(\theta) = \sum_x \frac{kP_n(x)}{P_\theta(x) + kP_n(x)}(P_d(x) - P_\theta(x))\frac{\partial}{\partial\theta}\log P_\theta(x).$$

  - This is a pointwise reweighting of the ML gradient.
- In fact, **as $k \to \infty$, the NCE gradient converges to the ML gradient**.
- If the noise distribution is non-zero *everywhere* and $P_\theta(x)$ is unconstrained, $P_\theta(x) = P_d(x)$ is the only optimum.
- If the model class does not contain $P_d(x)$, the location of the optimum depends on P_n.

NCE for training neural language models

- A neural language model specifies a large collection of distributions.
  - One distribution per context.
  - These distributions share parameters.
- We train the model by optimizing the sum of per-context NCE objectives weighted by the empirical context probabilities.
- If $P_\theta^h(w)$ is the probability of word w in context h under the model, the NCE objective for context h is

$$J_h(\theta) = E_{P_d^h}\left[\log \frac{P_\theta^h(w)}{P_\theta^h(w) + kP_n(w)}\right] + kE_{P_n}\left[\log \frac{kP_n(w)}{P_\theta^h(w) + kP_n(w)}\right].$$

- The overall objective is $J(\theta) = \sum_h P(h) J_h(\theta)$, where $P(h)$ is the empirical probability of context h.

The speedup due to using NCE

- The NCE parameter update is $\frac{cd+v}{cd+k}$ times faster than the ML update.
  - c is the context size
  - d is the representation dimensionality
  - v is the vocabulary size
  - k is the number of noise samples
- Using diagonal context matrices increases the speedup to $\frac{c+v}{c+k}$.

Practicalities

- NCE learns a different normalizing parameter for each context present in the training set.
  - For large context sizes and datasets the number of such parameters can get very large.
  - Fortunately, **learning works just as well if the normalizing parameters are fixed to 1**.
- When evaluating the model, the model distributions are normalized explicitly.
- Noise distribution: a unigram model estimated from the training data.
  - Use several noise samples per datapoint.
  - Generate new noise samples before each parameter update.

Penn Treebank results

- **Model**: LBL model with 100D feature vectors and a 2-word context.
- **Dataset**: Penn Treebank – news stories from Wall Street Journal.
  - Training set: 930K words
  - Validation set: 74K words
  - Test set: 82K words
  - Vocabulary: 10K words
- Models are evaluated based on their test set perplexity.
  - Perplexity is the geometric average of $\frac{1}{P(w|h)}$.
  - The perplexity of a uniform distribution over N values is N.

Results: varying the number of noise samples

Training algorithm	Number of samples	Test PPL	Training time (h)
ML		163.5	21
NCE	1	192.5	1.5
NCE	5	172.6	1.5
NCE	25	163.1	1.5
NCE	100	159.1	1.5

- NCE training is **14 times faster** than ML training in this setup.
- The number of samples has little effect on the training time because the cost of computing the predicted representation dominates the cost of the NCE-specific computations.

Results: the effect of the noise distribution

NUMBER OF SAMPLES	PPL USING UNIGRAM NOISE	PPL USING UNIFORM NOISE
1	192.5	291.0
5	172.6	233.7
25	163.1	195.1
100	159.1	173.2

- The empirical unigram distribution works much better than the uniform distribution for generating noise samples.
- As the number of noise samples increases the choice of the noise distribution becomes less important.

Application: MSR Sentence Completion Challenge

- **Large-scale application**: MSR Sentence Completion Challenge
- **Task**: given a sentence with a missing word, find the correct completion from a list of candidate words.
  - Test set: 1,040 sentences from five Sherlock Holmes novels
  - Training data:
    - 522 19th-century novels from Project Gutenberg (48M words)
- Five candidate completions per sentence.
  - Random guessing gives 20% accuracy.

Sample questions

- The stage lost a fine _____, even as science lost an acute reasoner, when he became a specialist in crime.
    a) linguist
    b) hunter
    c) actor
    d) estate
    e) horseman
- During two years I have had three _____ and one small job, and that is absolutely all that my profession has brought me.
    a) cheers
    b) jackets
    c) crackers
    d) fishes
    e) consultations

Question generation process (MSR)

- Automatic candidate generation:
  1. Pick a sentence with an infrequent target word (frequency < 10^{-4})
  2. Sample 150 unique infrequent candidates for replacing the target word from an LM with a context of size 2.
  3. If the correct completion scores lower than any of the candidates discard the sentence.
  4. Compute the probability of the word after the candidate using the LM and keep the 30 highest-scoring completions.
- Human judges pick the top 4 completions using the following guidelines:
  1. Discard grammatically incorrect sentences.
  2. The correct completion should be clearly better than the alternatives.
  3. Prefer alternatives that require “some thought” to answer correctly.
  4. Prefer alternatives that “require understanding properties of entities that are mentioned in the sentence”.

LBL for sentence completion

- We used LBL models with two extensions:
  - Diagonal context matrices for better scalability w.r.t word representation dimensionality.
  - Separate representation tables for context words and the next word.
- Handling sentence boundaries:
  - Use a special “out-of-sentence” token for words in context positions outside of the sentence containing the word being predicted.
- Word representation dimensionality: 100, 200, or 300.
- Context size: 2-10.
- Training time (48M words, 80K vocabulary): **1-2 days** on a single core.
  - **Estimated ML training time: 1-2 months**.

Sentence completion results

METHOD	CONTEXT SIZE	LATENT DIM	TEST PPL	PERCENT CORRECT
CHANCE	0			20.0
3-GRAM	2		130.8	36.0
4-GRAM	3		122.1	39.1
5-GRAM	4		121.5	38.7
6-GRAM	5		121.7	38.4
LSA	SENTENCE	300		49
RNN	SENTENCE	?	?	45
LBL	2	100	145.5	41.5
LBL	3	100	135.6	45.1
LBL	5	100	129.8	49.3
LBL	10	100	124.0	50.0
LBL	10	200	117.7	52.8
LBL	10	300	116.4	**54.7**
LBL	10×2	100	38.6	44.5

- LBL with a 10-word context and 300D word feature vectors sets a new accuracy record for the dataset.

Conclusions

- Noise-contrastive estimation provides a fast and simple way of training neural language models:
  - Over an order of magnitude faster than maximum-likelihood estimation.
  - Very stable even when using one noise sample per datapoint.
  - Models trained using NCE with 25 noise samples per datapoint perform as well as the ML-trained ones.
- Large LBL models trained with NCE achieve state-of-the-art performance on the MSR Sentence Completion Challenge dataset.