Efficient Exact Gradient Update for Training Deep Networks with Very Large Sparse Targets

Pascal Vincent ∗ Alexandre de Brébisson Xavier Bouthillier

Abstract

An important source of problems in training deep neural networks is the sparsity induced by very large vocabularies. When working naturally in e.g. neural language models or the learning of word-embeddings, often posed as predicting the probability of next words among a vocabulary of size D (e.g. 500,000). Computed the sparsity at layer, but typically not across-dimension output vectors from a last hidden layer of reasonable dimensionality (e.g. 500) incurs a prohibitive O(Dd) computational cost for such example, as doing updating the d x d output sparse matrices and computing the gradient for backpropagation for previous layers. While efficient handling of large sparse network inputs is trivial, this case of large sparse outputs is non-trivial, and thus far not well-handled with approximations such as blockwise softmax or sampling. We present an exact alternative. We can develop an elegant algorithmic approach for a family of loss functions that includes squared error and spherical softmax, can compute the cross-layer gradient update for the output weights, and gradient for backpropagation, all in O(d) per example instead of O(Dd) remarkably without ever computing the O(D-dimensional).

The Problem

Training deep neural networks with very large sparse targets is an important problem

- An example of Natural Language Models (e.g. D = 500,000 for one-hot targets).
- Efficient handling of large sparse inputs is trivial.
- But backpropagation with large sparse outputs is prohibitively expensive.
- Focus on output-layer maps; last hidden layer represented of reasonable dimensionality (e.g. 500) to very large output dimension (e.g. D = 500,000) with a few parameters (e.g. 5).

The Proposed approach

We can do much better than O(Dd). We can compute:
- Loss L
- gradient w.r.t. last hidden layer ∇L
- compute some gradient update to V
- all in O(d) without ever computing full output = VH!

First trick: ∇L and ∇V can be computed efficiently if we keep an up-to-date δ of matrix Q = WTV

Computing loss L

δL = ∑j(WhhVUyj − δhLj)

Computing gradient ∇V = last hidden layer ∇L

∇V = ∑kWhWTVVUyk

Practically speaking: we maintain up-to-date ∇L = VW (admissible cheaply)

Second trick: represent W implicitly as factorization W = QTV and update U and V instead

A factorized output layer

Naive gradient update is a one-step update to W

Wnew = W − 2(1/δ)QTVU

Regularity is decomposed in 2 sequential steps:

1. Wnew − W = 2(1/δ)QTVU
2. Wnew − W = 2(1/δ)QTVU

Bookkeeping operations as we update U and V

- Using factorization of W leaves intact the complexity of the computation of L and U.
- Need to maintain an up-to-date δ following rank 1 update to U.
- Admitted in (20) through Sherman-Morrison formula.
- Need to maintain an up-to-date δ following updates to U and V.
- Admitted in (19) through

Qnew = Q − UV / (δUV)

we suppose K < d < D

Detailed algorithm, benefits and limitations

Full algorithm (unilinear version)

Model size

Computational complexity

Number of multiply-adds

Speedup over naive computation

Anticipated benefits:
- Computation: O(d) vs. O(Dd)
- Speedup of δUV
- Memory access: for each example access only K d elements of V and d elements of U.
- Computational simplicity.

Limitations:
- Approach limited to a few functions expressible using (lll) and the matrix inverse.
- Not regular linear layers.
- Computational simplicity.
- Cost can still grow too fast in training.
- More typically apply numerical stabilization strategy.

Experimental validation

Training and testing of output layer computations: (top-to-bottom) Both naive backpropagation (unrealistic) and the proposed factorized parameter version learns the same target W. Efficiency with co = 0.670, on 315 billion and from TrueRank 0.687.

Evaluation of Stochastic 0990 as a function of time for full multi-layer neural language model with vocabulary of 100 billion.

Conclusion and future work

- We developed an original algorithm that yields a huge speedup for performing full exact gradient updates on networks with very large vocabularies.
- Gain in an algorithmic computational complexity improvement, not from low-level hardware specific tricks or tuning.
- Provide a natural sparse loss function, coming with standard implementations. Developing compiling application.