# Supplementary Materials for Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations

**Ting Chen** [1] **Martin Renqiang Min** [2] **Yizhou Sun** [1]

## 1. Proofs of Lemmas and Propositions

**Lemma 1.** *The number of embedding parameters used in KD encoding is $O(\frac{K}{\log K} d' \log N + C)$, where $C$ is the number of parameters of neural nets.*

*Proof.* As mentioned, the embedding parameters include code embedding matrix $\{\mathcal{W}\}$ and embedding transformation function $\theta_e$. There are $O(\frac{K}{\log K} \log N)$ code embedding vectors with $d'$ dimensions. As for the number of parameters in embedding transformation function such as neural networks (LSTM) $C$ that is in $O(d'^2)$, it can be treated as a constant to the number of symbols since $d'$ is independent of $N$, provided that there are certain structures presented in the symbol embeddings. For example, if we assume all the symbol embeddings are within $\epsilon$-ball of a finite number of centroids in $d$-dimensional space, it should only require a constant $C$ to achieve $\epsilon$-distance error bound, regardless of the vocabulary size, since the neural networks just have to memorize the finite centroids. □

.

**Proposition 1.** *A linear composition function $\boldsymbol{f}$ with no hidden layer is equivalent to a sparse binary low-rank factorization of the embedding matrix.*

*Proof sketch.* First consider when $K = 2$, and the composed embedding matrix can be written as $U = BC$, where $B$ is the binary code for each symbol, and $C$ is the code embedding matrix. This is a low rank factorization of the embedding matrix with binary code $B$. When we increase $K$, by representing a choice of $K$ as one-hot vector of size $K$, we still have $U = BC$ with additional constraints in B that it is a concatenation of $D$ one-hot vector. Due to the one-hot constraint, each row in $B$ will be sparse as only $1/K$ ratio of entries are non-zero, thus corresponds to a sparse binary low-rank factorization of the embedding matrix.

As the linear composition with no hidden layer can be limited in some cases as the expressiveness of the function highly relies on the number of bases or rank of the factorization. Hence, the non-linear composition may be more appealing in some cases.

**Proposition 2.** *Given the same dimensionality of the "KD code", i.e. K, D, and code embedding dimension $d'$, the non-linear embedding transformation functions can reconstruct the embedding matrix with higher rank than the linear counterpart.*

*Proof sketch.* As shown above, in the linear case, we approximate the embedding by a low-rank factorization, $U = BC$. The rank will be constrained by the dimensionality of binary matrix $B$, i.e. $KD$. However, if we consider a nonlinear transformation function $f$, we will have $U = f(B, C)$. As long as that no two rows in $B$ and no two columns in $C$ are the same, i.e. every data point has its quite code and every code has its unique embedding vector, then the non-linear function $f$, such as a neural network with enough capacity, can approximate a matrix $U$ that has much higher rank, even full rank, than $KD$.

## 2. The LSTM Code Embedding Transformation Function

Here we present more details on the LSTM code embedding transformation function. Assuming the code embedding dimension is the same as the LSTM hidden dimension, the formulation is given as follows.

$$\begin{aligned}
\boldsymbol{t}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1} U_t + \boldsymbol{b}_t) \\
\boldsymbol{i}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1} U_i + \boldsymbol{b}_i) \\
\boldsymbol{o}_j &= \sigma(\mathcal{W}^j_{\boldsymbol{c}^j} + \boldsymbol{h}_{j-1} U_t + \boldsymbol{b}_t) \\
\boldsymbol{m}_j &= \boldsymbol{t}_j \circ \boldsymbol{m}_{j-1} + \boldsymbol{i}_j \circ \tanh(\mathcal{W}^j_{\boldsymbol{c}^j} + U_m \boldsymbol{h}_{j-1} + \boldsymbol{b}_m) \\
\boldsymbol{h}_j &= \boldsymbol{o}_j \circ \tanh(\boldsymbol{m}_j),
\end{aligned}$$

where $\sigma(\cdot)$ and $\tanh(\cdot)$ are, respectively, standard sigmoid and tanh activation functions. Please note that the symbol

[1]Department of Computer Science, University of California, Los Angeles [2]NEC Labs America. Correspondence to: Ting Chen <tingchen@cs.ucla.edu>.

*Proceedings of the $35^{th}$ International Conference on Machine Learning*, Stockholm, Sweden, PMLR 80, 2018. 

index $i$ is ignored for simplicity.

## 3. Examples and Applications

Our proposed task-specific end-to-end learned "KD Encoding" can be applied to any problem involving learning embeddings to reduce model size and increase efficiency. In the following, we list some typical examples and applications, for which detailed descriptions can be found in the supplementary material.

**Language Modeling** Language modeling is a fundamental problem in NLP, and it can be formulated as predicting the probability over a sequence of words. Models based on recurrent neural networks (RNN) with word embedding (Mikolov et al., 2010; Kim et al., 2016) achieve state-of-the-art results, so on which we will base our experiments. A RNN language model estimates the probability distribution of a sequence of words by modeling the conditional probability of each word given its preceding words,

$$P(w_0, ..., w_N) = P(w_0) \prod_{i=1}^{N} P(w_i | w_0, ..., w_{i-1}), \quad (1)$$

where $w_i$ is the $i$-th word in a vocabulary, and the conditional probability $P(w_i|w_0, ..., w_{i?1})$ can be naturally modeled by a softmax output at the $i$-th time step of the RNN. The RNN parameters and the word embeddings are model parameters of the language model.

**Text Classification** Text classification is another important problem in NLP with many different applications. In this problem, given a training set of documents with each containing a number of words and its target label, we learn the embedding representation of each word and a binary or multi-class classifier with a logistic or softmax output, predicting the labels of test documents with the same vocabulary as in the training set. To test the "KD Encoding" of word embedding on several typical text classification applications, we use several different types of datasets: Yahoo answer and AG news represent topic prediction, Yelp Polarity and Yelp Full represent sentiment analysis, while DBpedia represents ontology classification.

**Graph Convolutional Networks for Semi-Supervised Node Classification** In (Kipf & Welling, 2016), graph convolutional networks (GCN) are proposed for semi-supervised node classification on undirected graphs. In GCN, the matrix based on standard graph adjacency matrix with added self connections after normalization, $\hat{A}$, is used to approximate spectral graph convolutions. As a result, $ReLU(\hat{A}XW)$ defines a non-linear convolutional feature transformation on node embedding matrix $X$ with a projection matrix $W$ and non-linear activation function $ReLU$.

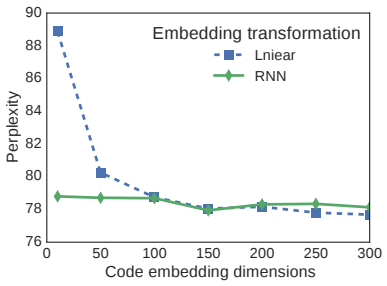


*Figure 1.* The perplexity on PTB as a function of different code embedding dimensions as well as the embedding transformation functions.

This layer-wise transformation can be repeated to build a deep network before making predictions using the final output layer. Minimizing a task-specific loss function, the network weights $W$s and the node embedding matrix $X$ are learned simultaneously using standard back-propagation. A simple GCN with one hidden layer takes the following form:

$$Z = f(X, A) = \text{softmax}(\hat{A}\text{ReLU}(\hat{A}XW_0)W_1), \quad (2)$$

where $W_0$ and $W_1$ are network weights, and softmax is performed in a row-wise manner. When the labels of only a subset of nodes are given, this framework is readily extended for graph-based semi-supervised node classification by minimizing the following loss function,

$$L = -\sum_{l=1}^{L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}, \quad (3)$$

where $L$ is the number of labeled graph nodes, $F$ is the total number of classes of the graph nodes, and $Y$ is a binary label matrix with each row summing to 1. We apply our proposed KD code learning to graph node embeddings in the above GCN framework for semi-supervised node classification.

**Hashing** The learned discrete code can also be seen as a data-dependent hashing for fast data retrieval. In this paper, we also perform some case studies evaluating the effectiveness of our learned KD code as hash code.

## 4. Additional Experimental Results

We also test the effects of different code embedding dimensions, and the result is presented in Figure 1. We found that linear encoder requires larger code embedding dimensionality, while the non-linear encoder can work well with related small ones. This again verifies the proposition 2.

Table 1 shows the effectiveness of variants of the tricks in continuous relaxation based optimization. We can clearly

*Table 1.* Effectiveness of different optimization tricks. Here, CR=Continuous Relaxation using softmax, STE=straight-through estimation, CDG=continuous distillation guidance.

| Variants | PPL |
|---|---|
| CR | 90.61 |
| CR + STE | 90.15 |
| CR + STE + temperature scheduling | 89.55 |
| CR + STE + entropy reg | 89.03 |
| CR + STE + entropy reg + PDG (w/o autoencod.) | 83.71 |
| CR + STE + entropy reg + PDG (w/ autoencod.) | 83.11 |

see that the positive impacts of temperature scheduling, and/or entropy regularization, as well as the auto-encoding. However, here the really big performance jump is brought by using the proposed distillation guidance.

## 5. Notations

For clarity, Table 1 provides explanations to major notations used in our paper.

*Table 2.* Notations

| Notations | Explanation |
|---|---|
| $\boldsymbol{c}$ | Codes. |
| $\boldsymbol{o}$ | One-hot representations of the code. |
| $\hat{\boldsymbol{o}}$ | Continuously relaxed $\boldsymbol{o}$. |
| $\boldsymbol{\pi}$ | code logits for computing $\hat{\boldsymbol{o}}$. |
| $\mathcal{W}$ | Code embedding matrix. |
| $\mathcal{T}$ | The transformation from symbol to the embedding , $\mathcal{T} = \boldsymbol{f} \circ \boldsymbol{\phi}$. |
| $\boldsymbol{\phi}$ | The transformation from symbol to code. |
| $\boldsymbol{f}$ | The code transformation function maps code to embedding. It has parameters $\theta = \{\mathcal{W}, \theta_e\}$ |
| $\boldsymbol{f}_e$ | The embedding transformation function maps code embedding vectors to a symbol embedding vector. |
| $\boldsymbol{v}$ | The composite symbol embedding vector. |
| $\Theta$ | The task-specific (non-embedding) parameters. |
| $\mathcal{U}$ | Pre-trained symbol embedding matrix. |
| $\boldsymbol{u}$ | Pre-trained symbol embedding vector. |
| $d$ | Symbol embedding dimensionality. |
| $d'$ | Code embedding dimensionality. |

## References

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pp. 2741–2749. AAAI Press, 2016.

Kipf, Thomas N and Welling, Max. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Mikolov, Tomáš, Karafiát, Martin, Burget, Lukáš, Černockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.