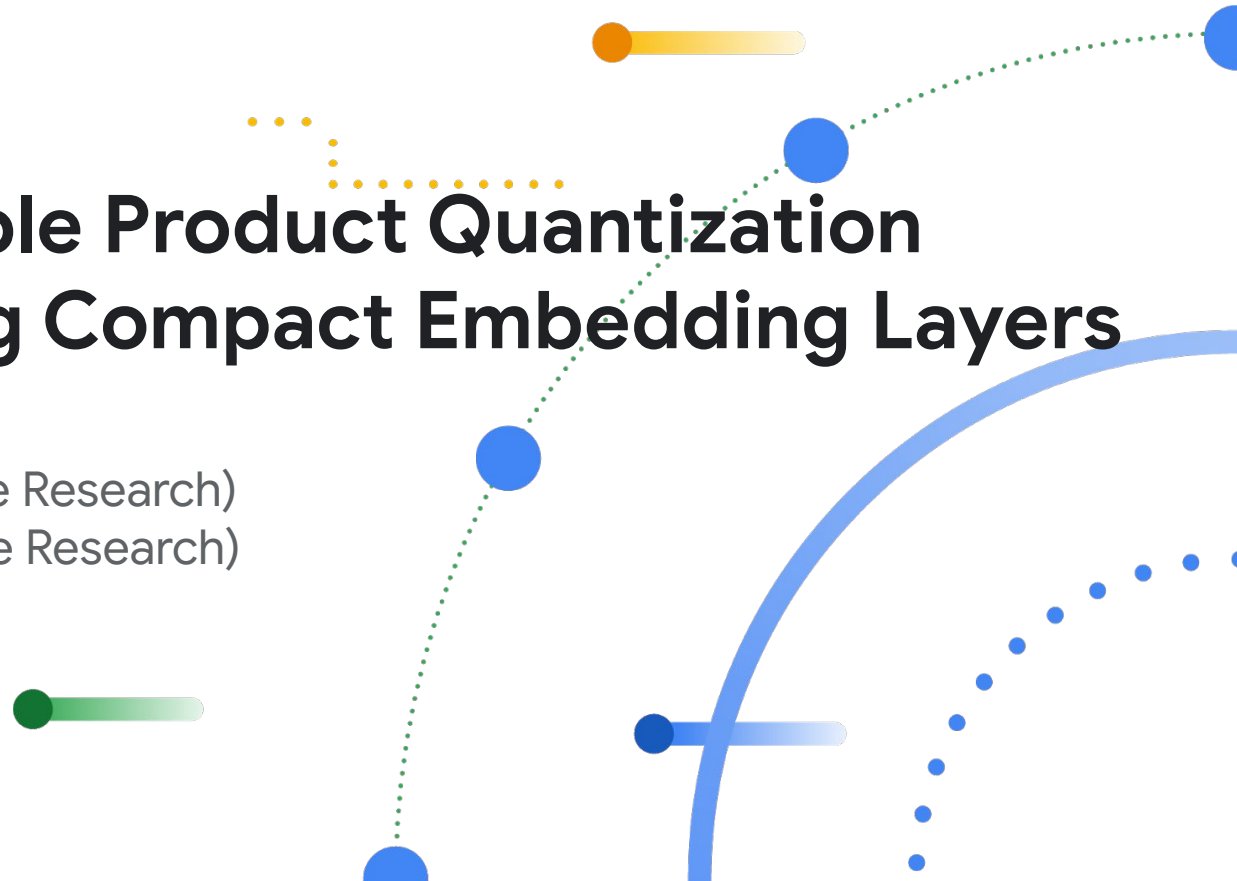
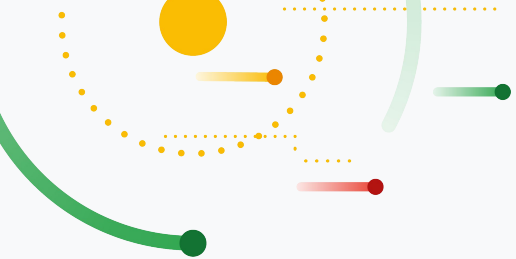
# Differentiable Product Quantization for Learning Compact Embedding Layers

Ting Chen (Google Research)
Lala Li (Google Research)
Yizhou Sun (UCLA)



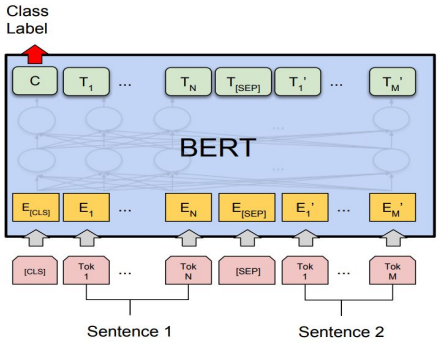


# What’s embedding?

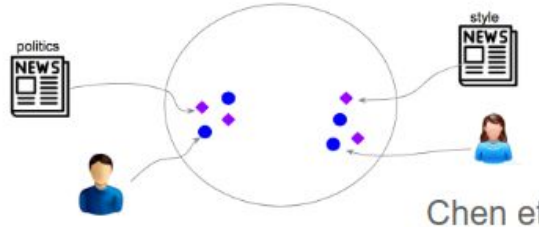
And why does it need to be compressed?

# Embedding: continuous vector representations of symbols

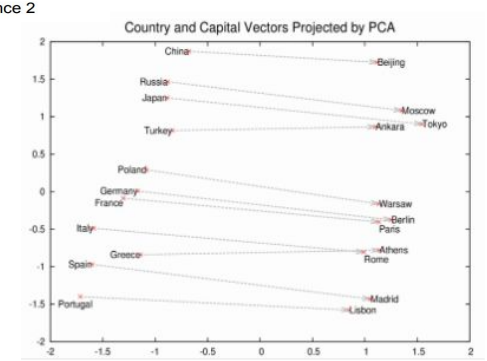
Embedding layer is commonly used in many NLP and other applications that deal with discrete symbols.



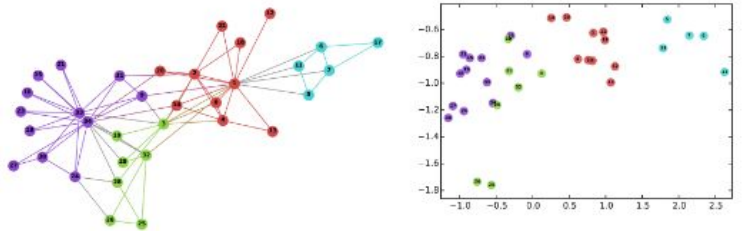

Devlin et al'18




Chen et al'17




Mikolov et al'13



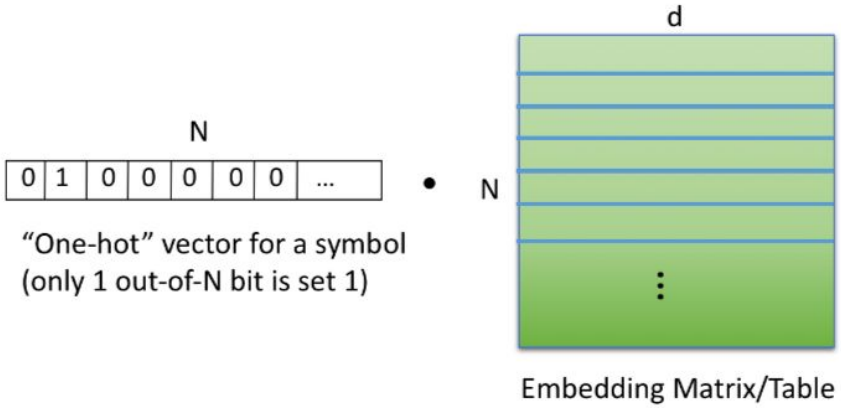

Perozzi et al'14

# Existing embedding approach

Maintains an embedding table where each row corresponds to a symbol, and use table lookup to retrieve the embedding.

This is equivalent to

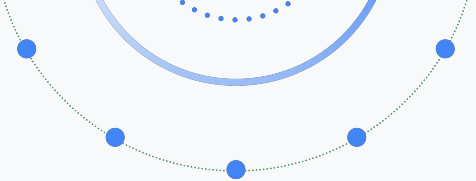
- Represent each symbol with a “one-hot” code vector
- Apply a linear/affine transformation on top



# Issues with the existing method

The number of parameters grows linearly with the number of symbols.

- Model size (physical storage).
- Redundancy
  - Lots of symbols with only a few observations (e.g. Zipf's law).
  - Many symbols share similar semantics (e.g. compositionality).



# How can we compress an embedding table?

In an End-to-end fashion.

# K-way D-dimensional Discrete Code

- The main idea is to compose embeddings from a small set of basis ones, by leveraging semantic similarities among symbols.
- Instead of using “one-hot” codes to encode symbols, we propose to use D-dimensional discrete codes, each dimension has K choices.


*Ting Chen, Martin Renqiang Min, and Yizhou Sun. "Learning K-way D-dimensional Discrete Codes for Compact Embedding Representations." ICML, 2018.*

# KD encoding framework

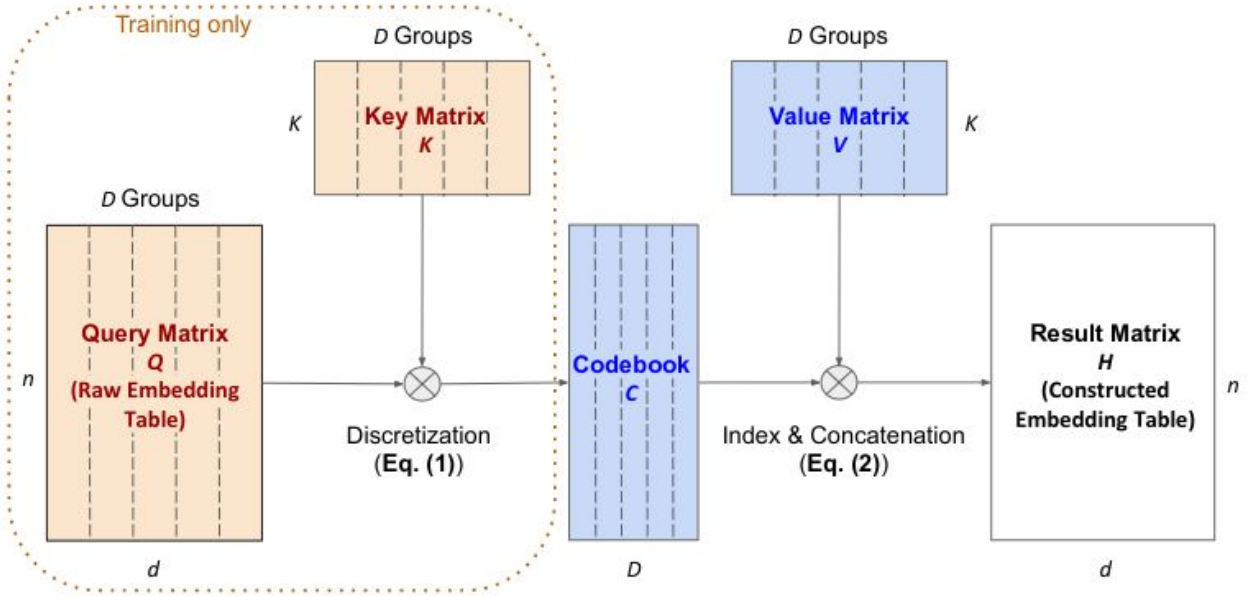
Instead of lookup the embedding vector in a table, we look up the codes, retrieve their embeddings, and use them to compose the final embedding vector.


(a) "one-hot" encoding (b) KD encoding

But, where do KD codes come from?

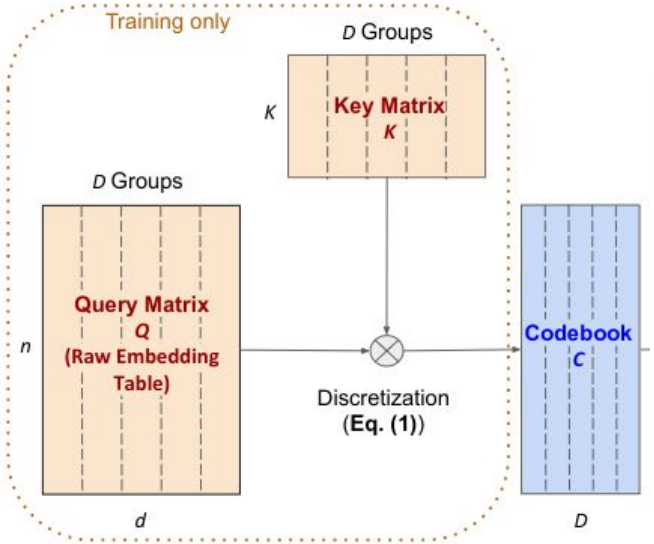
# The Proposed DPQ framework

DPQ compress a raw embedding Q into discrete codes C, and construct the final embedding matrix H.



# The Proposed DPQ framework

## Discretization function (training only)

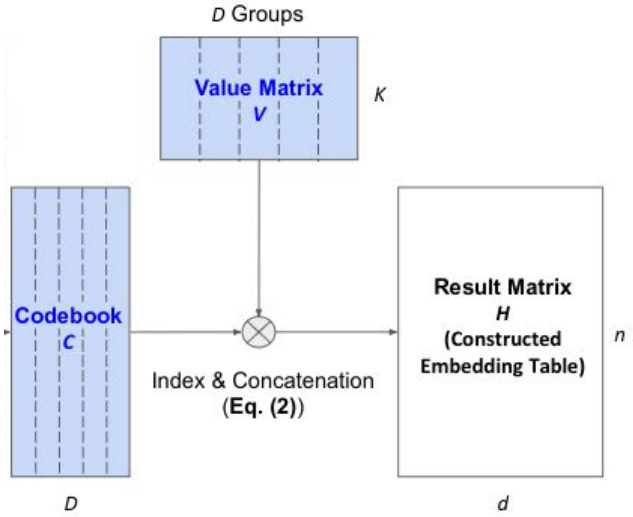



We divide the embedding space into multiple groups similar to product quantization. For each,

$$\mathbf{C}_i^{(j)} = \arg\min_k \operatorname{dist}\left(\mathbf{Q}_i^{(j)}, \mathbf{K}_k^{(j)}\right)$$

# The Proposed DPQ framework

## Reverse-discretization function




The final embedding is a concatenation of sub-vectors from each group.

$$\mathbf{H}_i = [\mathbf{V}^{(1)}_{\boldsymbol{c}_i^{(1)}}, \cdots, \mathbf{V}^{(j)}_{\boldsymbol{c}_i^{(j)}}, \cdots, \mathbf{V}^{(D)}_{\boldsymbol{c}_i^{(D)}}]$$

Inference time only needs code book and value matrix

**Algorithm 1** Inference of embedding for $i$-th token.

**Require:** $\mathbf{V} \in \mathbb{R}^{K \times D \times (d/D)}$, $\mathbf{C} \in \{1, ..., K\}^{n \times D}$

  **for** $j \in \{1, ..., D\}$ **do**

    $\boldsymbol{h}_i^{(j)} = \mathbf{V}^{(j)}_{\mathbf{C}_i^{(j)}}$

  **end for**

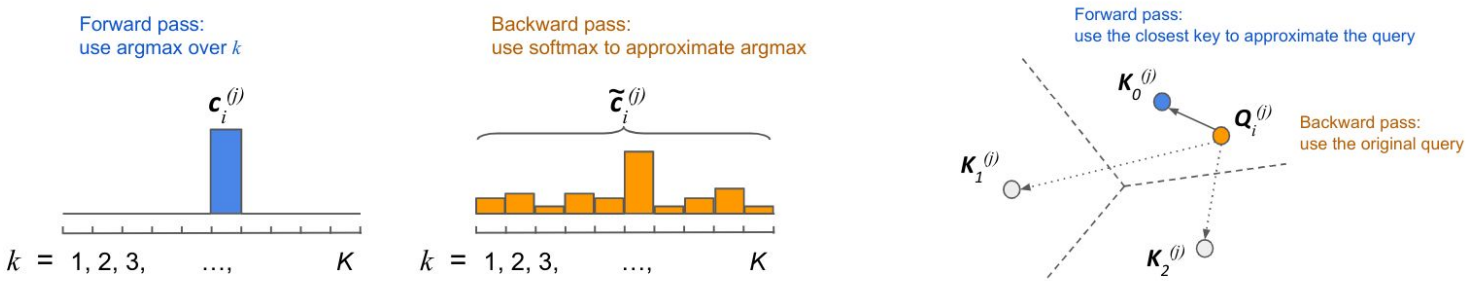
  **return** concatenate$(\boldsymbol{h}_i^{(1)}, \boldsymbol{h}_i^{(2)}, ..., \boldsymbol{h}_i^{(D)})$

# Two DPQ variants

Discretization function is not differentiable, we provide two approximations (variants):

DPQ-SX $\mathbf{C}_i^{(j)} = \arg\max_k \frac{\exp(\langle \mathbf{Q}_i^{(j)}, \mathbf{K}_k^{(j)} \rangle)}{\sum_{k'} \exp(\langle \mathbf{Q}_i^{(j)}, \mathbf{K}_{k'}^{(j)} \rangle)}$

DPQ-VQ $\mathbf{C}_i^{(j)} = \arg\min_k \|\mathbf{Q}_i^{(j)} - \mathbf{K}_k^{(j)}\|^2$




(a) Softmax-based (DPQ-SX) (b) Centroid-based (DPQ-VQ)

*Figure 2.* Illustration of two types of approximation to enable differentiability in DPQ.

# DPQ module is end2end differentiable

Main benefit: DPQ can be plugged into any differentiable neural network and learned end-to-end.

This means a few lines code change to use DPQ:

```
import full_embed
full_emb_layer = full_embed.FullEmbedding(vocab_size, emb_size)
output = full_emb_layer(inputs)
```

```
import dpq
dpq_layer = dpq.DPQEmbedding(vocab_size, emb_size,
                             k, d, dpq_variant, share_subspace)
output = dpq_layer(inputs)
```

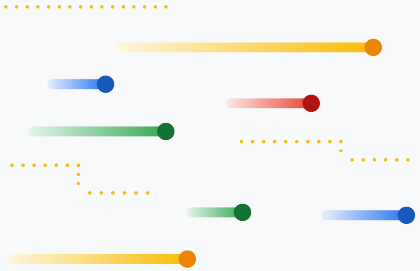
# Implementation Details

## Distance normalization.

Training with straight-through estimator can be unstable as the gradient is approximated. We normalize distance in a mini-batch so that each centroid will have a normalized distance distribution over batch samples.

## Subspace-sharing.

To further reduce parameters, one can share parameters among the D groups of Key/Value matrices.



# How well does DPQ work?

How much can we compress, without performance loss?

# 10 datasets across 3 language tasks

| Task | Dataset | Vocab Size | Tokenization | Base Model |
|---|---|---|---|---|
| LM | PTB<br>Wikitext-2 | 10,000<br>33,278 | Words | LSTM-based models from (Zaremba et al., 2014), three model sizes |
| NMT | IWSLT15 (En-Vi)<br>IWSLT15 (Vi-En) | 17,191<br>7,709 | Words | Seq2seq-based model from (Luong et al., 2017) |
| | WMT19 (En-De) | 32,000 | Sub-words | Transformer Base in (Vaswani et al., 2017) |
| TextC | AG News<br>Yahoo! Ans.<br>DBpedia<br>Yelp P<br>Yelp F | 69,322<br>477,522<br>612,530<br>246,739<br>268,414 | Words | One hidden layer after mean pooling of word vectors, similar to fastText from (Joulin et al., 2017) |

*Table 2.* Datasets and models used in our experiments. More details in Appendix B.

# Evaluation Metrics

Task performance metrics:

- Perplexity score for LM tasks
- BLEU score for NMT tasks
- Accuracy for text classification tasks.

Compression Ratio:

$$\text{CR} = \frac{\text{\# of bits used in the full embedding table}}{\text{\# of bits used in compressed model for inference}}$$

# Compared to full embedding baseline

We achieve 14-163X compression, without performance loss.

| Task | Metric | Dataset | Baseline | DPQ-SX | (Compr. Ratio↑) | DPQ-VQ | (Compr. Ratio↑) |
|---|---|---|---|---|---|---|---|
| LM | PPL↓ | PTB | 83.4 | **83.2** | **(163.2)** | 83.3 | (58.7) |
| | | Wikitext-2 | 95.6 | **95.0** | **(59.3)** | 95.9 | (95.3) |
| NMT | BLEU↑ | IWSLT15 (En-Vi) | **25.4** | 25.3 | (86.2) | 25.3 | (16.1) |
| | | IWSLT15 (Vi-En) | 23.0 | **23.1** | **(72.0)** | 22.5 | (14.1) |
| | | WMT19 (En-De) | **38.8** | **38.8** | **(18.0)** | 38.7 | (18.2) |
| TextC | Acc(%)↑ | AG News | **92.6** | 92.5 | (19.3) | 92.6 | (24.0) |
| | | Yahoo! Ans. | 69.4 | **69.6** | **(48.2)** | 69.2 | (19.2) |
| | | DBpedia | 98.1 | 98.1 | (24.1) | **98.1** | **(38.5)** |
| | | Yelp P | 93.9 | **94.2** | **(38.5)** | 93.9 | (24.0) |
| | | Yelp F | **60.3** | 60.1 | (48.2) | 60.2 | (24.1) |

*Table 3.* Comparisons of DPQ variants vs. the full embedding baseline on ten datasets across three tasks. We use ↓ to denote the lower the better, in contrast, ↑ means the higher the better.

# Compared to other compression techniques

- DPQ  is better than low-rank factorization.
- DPQ is better than post-training quantization.

| Method | PPL↓ | Compr. Ratio↑ |
|---|---|---|
| Full | 83.4 | 1.0 |
| Scalar quantization (8 bits) | 84.1 | 4.0 |
| Scalar quantization (6 bits) | 87.7 | 5.3 |
| Scalar quantization (4 bits) | 92.9 | 8.3 |
| Product quantization(64x325) | 84.0 | 8.3 |
| Product quantization(128x325) | 83.7 | 6.7 |
| Product quantization(256x325) | 83.7 | 5.3 |
| Low-rank (5X) | 84.8 | 5.0 |
| Low-rank (10X) | 85.5 | 10.2 |
| Ours (DPQ-VQ) | 83.3 | 58.7 |
| Ours (DPQ-SX) | **82.0** | **82.9** |

*Table 5.* Comparison of DPQ against traditional embedding compression techniques on the PTB LM task (medium-sized LSTM).

| Dataset | AG News | Yahoo! | DBPedia | Yelp P | Yelp F |
|---|---|---|---|---|---|
| Full | 92.6 (1.0) | 69.4 (1.0) | 98.1 (1.0) | 93.9 (1.0) | 60.3 (1.0) |
| Low-rank(10×) | 91.4 (10.4) | 69.5 (10.2) | 97.7 (10.3) | 92.4 (10.4) | 57.8 (10.3) |
| Low-rank(20×) | 91.5 (21.4) | 69.1 (21.5) | 97.9 (21.3) | 92.4 (21.5) | 57.3 (21.4) |
| (Chen et al., 2018b) | 91.6 (53.3) | 69.5 (31.7) | 98.0 (48.4) | 93.1 (48.6) | 59.0 (54.4) |
| DPQ-VQ | **92.6 (24.0)** | 69.2 (19.2) | **98.1 (38.5)** | 93.9 (24.0) | 60.2 (24.1) |
| DPQ-SX | 92.5 (19.3) | **69.6 (48.2)** | 98.1 (24.1) | **94.2 (38.5)** | **60.1 (48.2)** |

# Compared to other compression techniques

- DPQ provides better compression rates than other discrete code based approaches, and can be trained end-to-end in one-stage!

| Method | Small | | Medium | | Large | |
|---|---|---|---|---|---|---|
| | PPL↓ | CR↑ | PPL↓ | CR↑ | PPL↓ | CR↑ |
| Full | 114.5 | 1 | 83.4 | 1 | 78.7 | 1 |
| Shu'17 | 108.0 | 4.8 | 84.9 | 12.5 | 80.7 | 18.5 |
| Chen'18 | 108.5 | 4.8 | 89.0 | 12.5 | 86.4 | 18.5 |
| Chen'18+ | 107.8 | 4.8 | 83.1 | 12.5 | **77.7** | 18.5 |
| DPQ-SX | **105.8** | **85.5** | **82.0** | **82.9** | 78.5 | **238.3** |
| DPQ-VQ | 106.5 | 51.1 | 83.3 | 58.7 | 79.5 | **238.3** |

*Table 4.* Comparison of DPQ against recently proposed embedding compression techniques on the PTB LM task (LSTMs with three model sizes are studied). Metrics are perplexity (PPL) and compression ratio (CR).

# And, of course, BERT

- We simply replace BERT embedding table with ours, No hyper-parameter tuning.
- We pre-train BERT-base (Devlin et al., 2018) on 512-token sequences for 1M iterations with batch size 1024.

| Embeddings | CR | Squad 1.1 | Squad 2.0 | CoLA | MNLI | MRPC | XNLI |
|---|---|---|---|---|---|---|---|
| Full | 1.0 | 90.1± 0.1 /83.1± 0.3 | 78.8±0.6/75.5±0.6 | 80.6±0.7 | 84.3±0.1 | 85.9±0.5 | 53.5±0.4 |
| DPQ-SX | 37.0 | 90.0±0.1 /83.0±0.2 | 78.7±0.5/75.4±0.5 | 80.2±0.6 | 83.7±0.2 | 85.1±0.6 | 53.4±0.1 |

*Table 7.* Effect of using DPQ on BERT. DPQ gives a compression ratio of 37× on the embedding table while the model's performance on downstream tasks remains competitive.

# Conclusion

We present DPQ for end-to-end *embedding table compression*.

DPQ can serve as a drop-in replacement for existing embedding layer.

DPQ achieves 14-163X compression ratios of embedding table for a bunch of language tasks, at no or negligible performance cost.

*Code at: github.com/chentingpc/dpq_embedding_compression*