OGB-LSC: WIKIKG90MV2 TECHNICAL REPORT

Alberto Cattaneo[*]
Graphcore
albertoc@graphcore.ai

Daniel Justus[*]
Graphcore
danielj@graphcore.ai

Harry Mellor[*]
Graphcore
harrym@graphcore.ai

Douglas Orr[*]
Graphcore
douglaso@graphcore.ai

Jerome Maloberti
Graphcore

Zhenying Liu
Graphcore

Thorin Farnsworth
Graphcore

Andrew Fitzgibbon
Graphcore
awf@graphcore.ai

Blazej Banaszewski
Graphcore
blazejb@graphcore.ai

Carlo Luschi
Graphcore
carlo@graphcore.ai

November 2022

ABSTRACT

In this report we present the award-winning submission of our team wikiwiki to the WikiKG90Mv2 track of OGB-LSC@NeurIPS 2022. The task is link-prediction on the large-scale knowledge graph WikiKG90Mv2, consisting of 90M+ nodes and 600M+ edges. Our solution uses a diverse ensemble of 85 Knowledge Graph Embedding models combining five different scoring functions (TransE, TransH, RotatE, DistMult, ComplEx) and two different loss functions (log-sigmoid, sampled softmax cross-entropy). Each individual model is trained in parallel on a Graphcore Bow Pod_{16} using BESS (Balanced Entity Sampling and Sharing), a new distribution framework for KGE training and inference based on balanced collective communications between workers. Our final model achieves a validation MRR of 0.2922 and a test-challenge MRR of 0.2562, ranking among the top-3 submissions. The code is publicly available at: `https://github.com/graphcore/distributed-kge-poplar/tree/2022-ogb-submission`.

1 Introduction

Knowledge Graphs encode a knowledge base in the form of a heterogeneous directed graph, where facts are subject-predicate-object triples which are represented as labelled edges (relations) connecting pairs of nodes (entities). Over the past decades they have attracted growing interest, finding a wide variety of commercial applications ranging from question-answering (Hao et al., 2017) to recommender systems (Zhang et al., 2016). Knowledge Graph Embedding (KGE) models perform reasoning on knowledge graphs by learning a semantic-aware mapping of entities and relations to low-dimensional vector spaces V_e, V_r respectively, such that the plausibility of triples is measured by a scoring function of the head, relation and tail embeddings $f : V_e \times V_r \times V_e \to \mathbb{R}$. The learned embeddings can then be used to infer missing links in the graph (Knowledge Graph Completion) and for downstream tasks.

While the majority of the literature on KGE models focuses on relatively small graphs, real-world applications of commercial value increasingly require reasoning on graphs with hundreds of millions, or even billions, of entities and edges (Vrandecic & Krötzsch, 2014; Bollacker et al., 2008). It has therefore become paramount to investigate models with good scaling capabilities and develop effective distributed training frameworks running on multiple devices (Lerer et al., 2019; Zheng et al., 2020). KGE models are characterised by large memory requirements for storing

[*]Equal contribution

parameters (almost entirely concentrated in the embedding tables) with sparse memory access patterns, since at each training step only the embeddings of entities and relations in the mini-batch need to be accessed and updated. This makes parallelisation of KGE models potentially challenging, as communications between workers need to be carefully managed in order to keep embeddings synchronised without incurring in excessive overheads.

The *Open Graph Benchmark Large-Scale Challenge* (OGB-LSC) (Hu et al., 2021) aims to encourage the graph ML research community to work with realistically sized datasets and develop solutions able to meet real-world needs, by providing learning tasks with immediate applications on graphs at an unprecedented scale. The WikiKG90Mv2 track of the competition requires performing Knowledge Graph Completion on a graph with more than 90M entities. Our solution, which has achieved a top-3 ranking, consists of an ensemble of 85 KGE models combining a variety of well-established scoring functions (Bordes et al., 2013; Wang et al., 2014; Sun et al., 2019; Yang et al., 2015; Trouillon et al., 2016), implemented on the distributed processing framework BESS powered by Graphcore's Poplar SDK (Graphcore, 2022b) which allows for fast, communication-efficient training and inference (see Section 4).

2 Task and Dataset Description

The WikiKG90Mv2 dataset (Hu et al., 2021) is a knowledge graph constructed from the Wikidata open knowledge base (Vrandecic & Krötzsch, 2014). We denote by $\mathcal{E}$ the set of entities (Wikidata items) in the knowledge graph and by $\mathcal{R}$ the set of relations (Wikidata linking properties). A subject-predicate-object claim is then abstracted as a triple (h, r, t) with $h, t \in \mathcal{E}$ and $r \in \mathcal{R}$. The training set $\mathcal{T}$ consists of positive triples representing true facts in the knowledge base. A 768-dimensional feature embedding vector is also provided for each entity and relation, obtained by encoding the title and description of the corresponding Wikidata entry with MPNet (Song et al., 2020).

Table 1: WikiKG90Mv2 dataset

$\lvert\mathcal{E}\rvert$	91,230,610
$\lvert\mathcal{R}\rvert$	1387
$\lvert\mathcal{T}\rvert$	601,062,811
# validation queries	15,000
# test-dev queries	10,000
# test-challenge queries	10,000

The task is to impute missing links in the knowledge graph, by predicting the top-10 tail entities t which are most likely to complete a query $(h, r, ?)$. The metric used is the Mean Reciprocal Rank (MRR) of the ground-truth tail among the top-10 candidates (with a reciprocal rank of 0 if the ground-truth is not present in the set of predictions). The validation and test sets are extracted from snapshots of the Wikidata knowledge base at later time-stamps.

2.1 Dataset Exploration

Given the large number of nodes and edges in the knowledge graph, it is useful to compare statistics for the four dataset splits (training, validation, test-dev and test-challenge sets). Figure 1a highlights a striking discrepancy in the the distribution of relations between training and validation/test sets. While the same relation (ID 481) is the one appearing most frequently in all sets, it spans more than 40% of the training triples but only 3.3%, 5.8% and 5.9% of the triples in the validation, test-dev and test-challenge sets respectively. As detailed in the dataset documentation (Hu et al., 2022), the validation and tests sets have been sampled so that the final relation counts are proportional to the cube root of the counts in the respective Wikidata dumps. Sampling from the training set with a similar strategy produces a better distribution alignment (Figure 1a). When looking at the distribution of entities, we notice that only 33,179,325 of them (roughly one third of $|\mathcal{E}|$) appear as tails in the training set. As shown in Figure 1b, the cube root sampling strategy helps to mitigate the difference between the training and validation distributions of tails, however almost 20% of tail entities in the validation set are never used as tails in the training set.

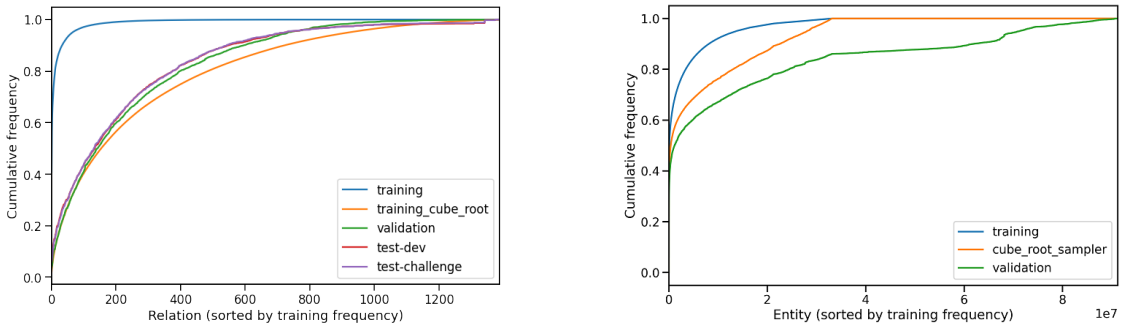
3 Methodology

3.1 Model Architecture

Encoder All KGE models in the final ensemble share the same shallow encoding strategy, which we describe in this paragraph. For an entity $e \in \mathcal{E}$, we denote by $\boldsymbol{e}_F \in \mathbb{R}^{768}$ its MPNet text features provided in the dataset and define a trainable entity embedding $\boldsymbol{e}_S \in \mathbb{R}^d$. We use linear layers $\mathbf{M}_H, \mathbf{M}_T \in \mathbb{R}^{d \times 768}$ to project $\boldsymbol{e}_F$ to $\mathbb{R}^d$ for head and tail entities respectively, optionally with $\mathbf{M}_H = \mathbf{M}_T$. The final entity embedding is given by:

$$\begin{aligned} \boldsymbol{e} &= \boldsymbol{e}_S + \mathbf{M}_H \boldsymbol{e}_F \qquad \text{for head entities;} \\ \boldsymbol{e} &= \boldsymbol{e}_S + \mathbf{M}_T \boldsymbol{e}_F \qquad \text{for tail entities.} \end{aligned} \tag{1}$$

Since the number of relations is small we do not make use of their text features, but only train a shallow embedding $\boldsymbol{r} \in \mathbb{R}^k$ for each $r \in \mathcal{R}$, where $k = d/2$ for RotatE and $k = d$ otherwise.



(a) (b)

Figure 1: (a) Cumulative frequencies of relations in the four dataset splits. The distribution of the cube root of relation counts in the training set is also displayed. (b) Cumulative frequencies of tail entities in the training and validation sets.

Scoring Functions The model's decoder assigns to each triple (h, r, t) a score $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) \in \mathbb{R}$, where $\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}$ are the embedding vectors for the head entity h, relation r and tail entity t respectively, obtained through the encoder as in eq. (1). We consider five different scoring functions: TransE (Bordes et al., 2013), TransH (Wang et al., 2014), RotatE (Sun et al., 2019), DistMult (Yang et al., 2015) and ComplEx (Trouillon et al., 2016) (see Table 2). For the three distance-based scoring functions, namely TransE, TransH and RotatE, we test both L^1 and L^2 distances. In the case of TransH, for each relation $r \in \mathcal{R}$ we have the additional trainable parameter given by $\boldsymbol{w}_r \in \mathbb{R}^d$, which represents the unit normal vector to the relation-specific hyperplane onto which the entity embeddings are projected.

Loss Functions Following standard convention, we optimise KGE models by imposing that the score $f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})$ of a positive triple $(h, r, t) \in \mathcal{T}$ is larger than the score of (pseudo)negative samples $(h, r, t'_i), i = 1, \ldots, N$, obtained by randomly replacing the tail entity t. Two different loss functions $\mathcal{L}$ are considered.

- **Log-sigmoid loss** (Sun et al., 2019).

$$\mathcal{L}(h, r, t) = -\log \sigma(\gamma + f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})) - \sum_{i=1}^{N} w_i \log \sigma(-\gamma - f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}'_i))$$

  where $\gamma > 0$ is a fixed margin for distance-based scoring functions ($\gamma = 0$ for DistMult and ComplEx), σ is the sigmoid function and we use self-adversarial negative sample weighting

$$w_i = \texttt{StopGrad}\left(\frac{e^{a \cdot f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}'_i)}}{\sum_{j=1}^{N} e^{a \cdot f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}'_j)}}\right)$$

  to upweight negative samples with higher scores (i.e. those which are more difficult for the current model to tell apart). Here $a \geq 0$ is a hyperparameter tuning the temperature of self-adversarial negative sampling.
- **Sampled softmax cross entropy loss** (Jean et al., 2015). A variant of plain softmax cross entropy loss which uses the target class and a set of N negative samples to estimate the log-sum-exp of logits over all possible

Table 2: Scoring functions and their ability to model four fundamental relation properties: S = Symmetry; AS = Antisymmetry; I = Inversion; C = Composition. For RotatE and ComplEx we assume d even and denote by $\mathbb{C}^{\frac{d}{2}}$ the vector space $\mathbb{R}^d = (\mathbb{R} \oplus i\mathbb{R})^{\frac{d}{2}}$ with the structure of $\mathbb{R}$-algebra induced by the product of complex numbers. $\circ$ denotes the Hadamard product; $p \in \{1, 2\}$.

Model	**Scoring function**		**S**	**AS**	**I**	**C**
TransE	$-\lVert \boldsymbol{h} + \boldsymbol{r} - \boldsymbol{t} \rVert_p$	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^d$	✗	✓	✓	✓
TransH	$-\lVert (\boldsymbol{h} - \boldsymbol{w}_r^\intercal \boldsymbol{h} \boldsymbol{w}_r) + \boldsymbol{r} - (\boldsymbol{t} - \boldsymbol{w}_r^\intercal \boldsymbol{t} \boldsymbol{w}_r) \rVert_p$	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{w}_r, \boldsymbol{t} \in \mathbb{R}^d$	✓	✓	✗	✗
RotatE	$-\lVert \boldsymbol{h} \circ e^{i\boldsymbol{r}} - \boldsymbol{t} \rVert_p$	$\boldsymbol{h}, \boldsymbol{t} \in \mathbb{C}^{\frac{d}{2}}, \boldsymbol{r} \in \mathbb{R}^{\frac{d}{2}}$	✓	✓	✓	✓
DistMult	$\langle \boldsymbol{r}, \boldsymbol{h}, \boldsymbol{t} \rangle$	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{R}^d$	✓	✗	✗	✗
ComplEx	$\mathrm{Re}\langle \boldsymbol{r}, \boldsymbol{h}, \bar{\boldsymbol{t}} \rangle$	$\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t} \in \mathbb{C}^{\frac{d}{2}}$	✓	✓	✓	✗

classes (in our case, the 90M+ entities in the knowledge graph). We can lower the variance of such estimator by separating the contribution to the log-sum-exp of the target class and introducing a correction $c = \log \frac{|\mathcal{E}|-1}{N}$ for the other terms as follows:

$$\mathcal{L}(h, r, t) = -f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) + \log \left(e^{f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t})} + \sum_{i=1}^{N} e^{f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}'_i)+c} \right).$$

Regularisation We regularise both losses with the L^3 norm of embedding vectors. This was motivated in (Lacroix et al., 2018) for tensor-decomposition scoring functions such as DistMult and ComplEx, however we find beneficial effects also with distance-based scores. We compute the L^3 norm on the final entity embedding in eq. (1) and its separate components, namely the trainable shallow embedding and the text feature projection. For a micro-batch $\mathcal{B}$ with (shared) negative tails $t'_i, i = 1, \ldots, N$, the regularisation term added to the micro-batch loss is $\lambda_T \Omega_T^{\mathcal{B}} + \lambda_S \Omega_S^{\mathcal{B}} + \lambda_F \Omega_F^{\mathcal{B}}$, where $\lambda_T, \lambda_S, \lambda_F$ are distinct regularisation parameters and

$$\Omega_T^{\mathcal{B}} = \sum_{(h,r,t)\in\mathcal{B}} (\|\boldsymbol{h}\|_3 + \|\boldsymbol{t}\|_3) + \sum_{i=1}^{N} \|\boldsymbol{t}'_i\|_3 ,$$

$$\Omega_S^{\mathcal{B}} = \sum_{(h,r,t)\in\mathcal{B}} (\|\boldsymbol{h}_S\|_3 + \|\boldsymbol{t}_S\|_3) + \sum_{i=1}^{N} \left\|\boldsymbol{t}'_{S,i}\right\|_3 ,$$

$$\Omega_F^{\mathcal{B}} = \sum_{(h,r,t)\in\mathcal{B}} (\|\mathbf{M}_H \boldsymbol{h}_F\|_3 + \|\mathbf{M}_T \boldsymbol{t}_F\|_3) + \sum_{i=1}^{N} \left\|\mathbf{M}_T \boldsymbol{t}'_{F,i}\right\|_3 .$$

As an additional form of regularisation, we also experiment with applying dropout (Srivastava et al., 2014) to the linear projections $\mathbf{M}_H \boldsymbol{e}_F$, $\mathbf{M}_T \boldsymbol{e}_F$ in eq. (1), before summing the output with $\boldsymbol{e}_S$.

3.2 Inference

Given a test query $(h, r, ?)$, inference is performed by traversing all entities in the knowledge graph and selecting the tails t that realise the top-K scores in $\{f(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}), t \in \mathcal{E}\}$. Despite this approach having time complexity $O(|\mathcal{E}| + |\mathcal{E}| \log K)$, our BESS distributed setup allows us to perform validation and testing fast enough to avoid any form of bias introduced by candidate selection methods (Chao et al., 2022).

Ensemble We use the following mean-ensembling strategy to combine the predictions of M trained individual models for a query $(h, r, ?)$. Let $\{t_1^m, \ldots, t_K^m\}$ be the top-K ranked set of tails predicted by the m-th model, for $K \geq 10$. For a power hyperparameter $p \neq 0$, we assign the following rank-based score to each entity $t \in \mathcal{E}$:

$$s(t) = \sum_{m=1}^{M} s_m(t), \qquad s_m(t) = \begin{cases} -\operatorname{sgn}(p) k^p & \text{if } t = t_k^m \\ -\mathbb{1}_{p>0} \cdot (K+1)^p & \text{if } t \notin \{t_1^m, \ldots, t_K^m\} \end{cases} \tag{2}$$

and select the entities with top-10 $s(t)$ values as final ranked predictions.

4 Acceleration and Distribution Strategy

The distribution scheme BESS (Balanced Entity Sampling and Sharing) involves a master process coordinating D workers (in our case, single Graphcore IPUs), with the key feature that workers can exchange data directly between them via collective communications, so that no additional parameter server is needed.

Partitioning Each embedding table is randomly partitioned row-wise across the D workers, in shards of equal sizes stored in the workers' memory. We denote by $\mathcal{E}_1, \ldots, \mathcal{E}_D$ the partitions of the set of entities $\mathcal{E}$, with $|\mathcal{E}_i| = \left\lceil \frac{|\mathcal{E}|}{D} \right\rceil$. This splitting induces a partitioning of the triples (h, r, t) in the knowledge graph based on the location of the head and tail entities: $\mathcal{T}_{i,j} = \{(h, r, t) \in \mathcal{T} : h \in \mathcal{E}_i, t \in \mathcal{E}_j\}$, $i, j = 1, \ldots, D$. Even with random partitioning of entities, the size variance of the D^2 partitions $\mathcal{T}_{i,j}$ will depend on the connectivity patterns of the specific knowledge graph; for WikiKG90Mv2 we find them to be always sufficiently well-balanced.

Since the number of relations in knowledge graphs is typically small (compared to the number of entities), we can afford to use an AllGather collective to reconstruct the whole relation embedding table on each worker before extracting the relevant embeddings needed to compute the micro-batch loss or predictions. The same strategy is used to split and retrieve the head and tail feature projection matrices $\mathbf{M}_H, \mathbf{M}_T$ and all weights other than entity embeddings, whose sharing requires an ad hoc strategy that is different for training and inference.

Training At each training step, the master process samples a micro-batch $\mathcal{B}_i$ for each of the workers, with $|\mathcal{B}_i| = B \equiv 0 \pmod{D}$ for $i = 1, \dots, D$. All triples $(h, r, t) \in \mathcal{B}_i$ have $h \in \mathcal{E}_i$, while tail entities are equally distributed across partitions $\mathcal{E}_j$, i.e.

$$\mathcal{B}_i = \bigcup_{j=1}^{D} \mathcal{B}_{i,j}, \qquad \mathcal{B}_{i,j} \subset \mathcal{T}_{i,j},\ |\mathcal{B}_{i,j}| = \frac{B}{D}. \tag{3}$$

Triples $(h, r, t) \in \mathcal{B}_{i,j}$ are sampled (with replacement) from $\mathcal{T}_{i,j}$ according to the following probability distribution:

$$p((h, r, t)) = p((h, r, t)|r)p(r) = \frac{1}{n_r} \frac{\sqrt[3]{n_r}}{\sum_{r' \in \mathcal{R}} \sqrt[3]{n_{r'}}}$$

where, for a relation $r' \in \mathcal{R}$, we denote $n_{r'} := |\{(h, r, t) \in \mathcal{T}_{i,j} : r = r'\}|$. As motivated in Section 2.1, we force the distribution of relations produced by the sampler to be proportional to the cube root of relation frequencies in the training set, in order to better align it with the validation and test sets, thus reducing distribution shift.

Together with positive triples, the master process also samples sets of entities to construct negative samples. We adopt *negative sample sharing*, i.e. use the same set of corrupted tails for all triples in a micro-batch. This allows us to increase the effective negative sample size without increasing communication costs, while also reducing the computational cost of scoring negative triples (as negative tail embeddings can be broadcasted across the micro-batch). The set of negative tails used for the micro-batch $\mathcal{B}_i$ is given by

$$\mathcal{N}_i = \bigcup_{j=1}^{D} \mathcal{N}_{i,j}, \qquad \mathcal{N}_{i,j} \subset \mathcal{E}_j,\ |\mathcal{N}_{i,j}| = \frac{N}{D} \tag{4}$$

where $N \equiv 0 \pmod{D}$ is the total number of negative samples for each positive triple.

The micro-batch and negative sample structures used by BESS (eqs. (3) and (4)) present three main advantages. Firstly, the fact that each micro-batch uses entities coming from all partitions $\mathcal{E}_j$, both for positive and negative triples, mitigates a potential source of bias and ensures a variety which is beneficial to the final embedding quality (Kochsiek & Gemulla, 2021). Secondly, as $\mathcal{B}_i$ is processed on worker i (which stores the embeddings for entities $\mathcal{E}_i$) only tail embeddings (positive and negative) need to be communicated between workers. Thirdly, by taking an equal number of triples from each partition $\mathcal{T}_{i,j}$ and of corrupted tails from each $\mathcal{E}_j$ we can efficiently organise the embedding sharing by means of AllToAll collectives, as every pair of workers needs to exchange the same amount of data. More specifically, the data sent from worker j to worker i consists of the embeddings of the B/D tail entities in $\mathcal{B}_{i,j}$ and the N/D entities in $\mathcal{N}_{i,j}$. This also implies that communication costs are constant across training steps and every worker performs the same amount of work, so that – even with frequent synchronisations – no significant idle time is introduced.

Inference Different communication patterns are required at inference time, where a query $(h, r, ?)$ needs to be scored against all tails $t \in \mathcal{E}$. A micro-batch of queries $\mathcal{Q}_i = \{(h, r) : h \in \mathcal{E}_i\}$ is fed by the master process to worker $i = 1, \dots, D$. The relevant head entities are gathered from local memory and then shared through an AllGather collective between all devices. Worker i proceeds to score the queries $(h, r) \in \bigcup_{j=1}^{D} \mathcal{Q}_j$ against all local tails $t \in \mathcal{E}_i$ and returns the top-K predictions (with the corresponding scores) to the host, where for each query a final top-K reduction is performed on the $D \cdot K$ retrieved scores in order to select the model's set of predictions.

4.1 Hardware Considerations

Training performance depends directly on computation and communication costs, and indirectly on achievable batch size within a memory limit. Up to small constant relative factors, the time taken for computing a single training step is

$$\begin{aligned} t_{\text{compute}} &= c_{\text{compute}} \cdot (B \cdot N \cdot d + (B + N) \cdot |\boldsymbol{e}_F| \cdot d), \\ t_{\text{comms}} &= c_{\text{comms}} \cdot (B + N) \cdot (d + |\boldsymbol{e}_F|), \end{aligned}$$

with local memory usage

$$(B + N) \cdot (d + |\boldsymbol{e}_F|) < L,$$

where c_{compute}, c_{comms} and L are hardware-specific constants. If we assume d and $|\boldsymbol{e}_F|$ are fixed, the amount of useful work done in a training step is proportional to $B \cdot N$. Efficient training therefore requires large B and N, within the limit imposed by local memory. A hardware platform for efficient training requires low c_{compute} or high achieved FLOP/s[2]. It also requires sufficiently low c_{comms} or high memory bandwidth (byte/s), and high L or large local memory (bytes), although these can be traded off against each other.

Our training system uses a single Bow Pod_{16}, providing 16 IPUs each with 32 GiB streaming memory, 900 MiB in-processor memory and 350 TFLOP/s compute in FP16 precision (Graphcore, 2022a). IPUs are connected in a 2D torus by high-speed IPU-Links giving a total bidirectional bandwidth of 320 GiB/s between a chip and its peers. We designate each IPU as a worker ($D = 16$) and reserve the entire streaming memory to store partitioned entity embeddings, associated optimiser state and features. To save memory and bandwidth these are stored in FP16. In-processor memory is used as a permanent store for all other parameters and optimiser state, for code and as working memory.

This configuration supports a maximum entity embedding size $d = 416$, where for each entity its embedding, optimiser state and features are packed into a 4 kiB row in streaming memory. Maximum micro-batch size B and negative sample size N depend on scoring function and d, for example $\{\text{TransE}, d = 256, B = 512, N = 1536\}$. In this example configuration, a single training step takes 6.5 ms, giving throughput $1.26 \cdot 10^6$ triples/s for an epoch time of 8 minutes. Inference uses a micro-batch size $|\mathcal{Q}_i| = 128$ to compute top-100 predictions for the 15,000 validation samples in 102 seconds.

The Bow Pod_{16} hardware platform and Poplar software stack provide fine-grained control over on-device computation and access to streaming memory, enabling effective use of in-processor memory to achieve large batch size. This allows for reasonably efficient operation with the available memory communication bandwidth.

5 Experimental Setup

Models have been trained on a Bow Pod_{16} with a micro-batch size of 256-640 (per IPU) for $5 \cdot 10^6$ steps, corresponding to 34-85 epochs. The MRR was evaluated periodically during training on 15,000 training samples and on the validation set. Hyperparameter settings for the different scoring functions can be found in Table 3[3]. All scoring functions have been trained in combination with both log-sigmoid loss and sampled softmax cross entropy loss. TransE, TransH, and RotatE models have been trained with L^1 and L^2 distances. For the majority of models the learning rate has been decayed linearly to zero over the course of training.

Table 3: Typical model configurations.

Scoring function	Initial learning rate	Global batch size	Embedding size
TransE	$[5 \cdot 10^{-4}, 3 \cdot 10^{-3}]$	$\{4096, 7168, 8192\}$	$\{256, 384\}$
TransH	$[2 \cdot 10^{-4}, 4 \cdot 10^{-3}]$	4096	256
RotatE	$[5 \cdot 10^{-4}, 10^{-2}]$	$\{4096, 7168\}$	256
DistMult	$[10^{-3}, 6 \cdot 10^{-3}]$	$\{4096, 8192\}$	$\{256, 384, 400\}$
ComplEx	$[5 \cdot 10^{-4}, 5 \cdot 10^{-3}]$	$\{4096, 8192, 10240\}$	400

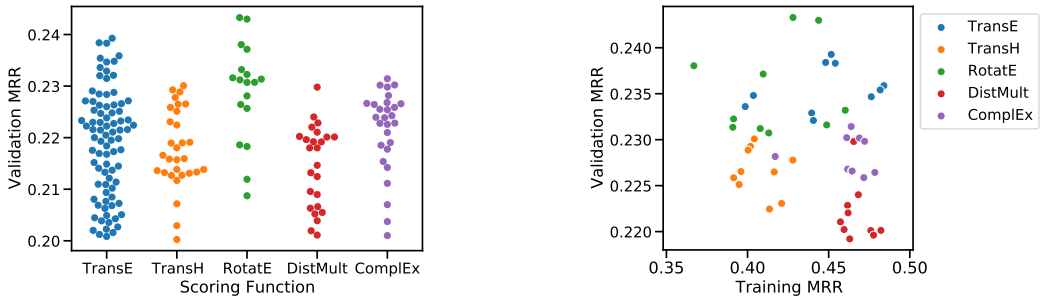
6 Results

6.1 Individual Models

To achieve the best possible MRR we aimed at maximising the diversity of models in our ensemble. This approach can be expected to benefit from complementary properties of different models, such as the properties of scoring functions specified in Table 2 and their different capabilities to model one-to-one or many-to-one relations. We trained a total of 259 models to completion with different scoring functions (and distances), loss functions and sets of hyperparameters. Out of these models, 185 achieved a validation MRR > 0.2 (Figure 2a).

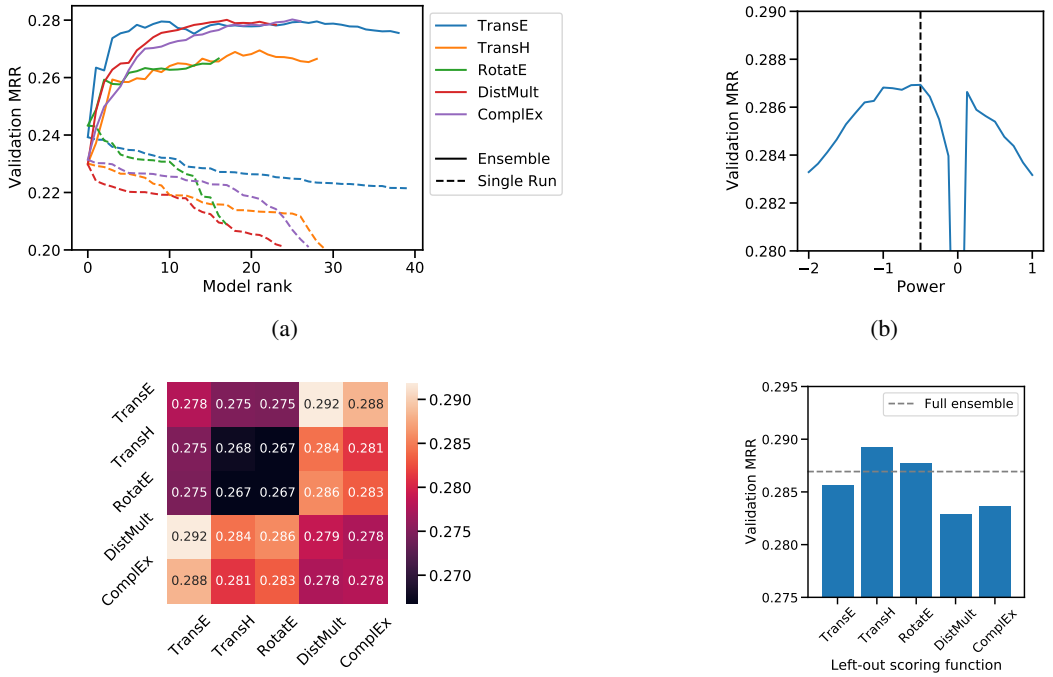
[2]FLOP/s: floating-point operations per second

[3]Detailed information on the hyperparameter settings and accuracy of all models used in the final ensemble can be found at `https://github.com/graphcore/distributed-kge-poplar/tree/resources/2022-ogb-submission`



(a) (b)

Figure 2: (a) Validation MRRs of the 185 models (84 TransE, 30 TransH, 28 ComplEx, 25 DistMult, 18 RotatE) that have been trained to a validation MRR of at least 0.2. (b) Validation MRR of the 10 best models per scoring function plotted against their respective training MRR.



(a) (b)

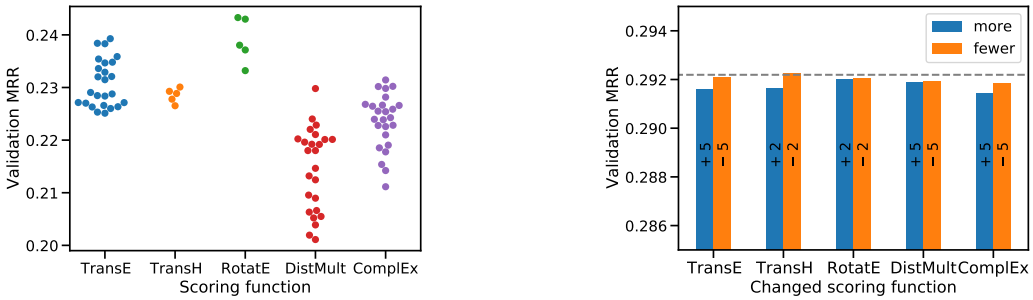
(c) (d)

Figure 3: (a) Validation MRR of the k-th best individual model (dashed) and of the ensemble of the k best models (solid) per scoring function. (b) The effect of the power hyperparameter p in eq. (2) on validation MRR for an ensemble of 50 models (the best 10 for each scoring function). (c) Validation MRR of the ensemble of the 20 best models per scoring function (main diagonal) and the $10 + 10$ best models of two scoring functions (off-diagonal). (d) Ablation of the contribution of different scoring functions to the ensemble of the 10 best models for each scoring function.

Depending on the scoring function used by the model, a different tendency to overfit on the training data can be observed. In particular, models using DistMult or ComplEx reach a substantially higher MRR on the subsample of the training set than on the validation set (Figure 2b).

6.2 Ensemble

Using the mean-ensembling strategy laid out in eq. (2) to create an ensemble, powers $p \in [-1.0, -0.5]$ yield good results (Figure 3b). As relying less on few top results intuitively generalises better, we selected $p = -0.5$ for the final ensemble.



(a) (b)

Figure 4: (a) 85 models (25 TransE, 5 TransH, 5 RotatE, 25 DistMult, 25 ComplEx) have been selected for the final ensemble based on their validation MRR. (b) Increasing (blue) or reducing (orange) the number of models per scoring function does not substantially improve validation MRR of the ensemble.

Depending on the scoring function, models benefit to a different degree from ensembling: although the best individual models use RotatE (Figure 2a), ensembles of a single scoring function among TransE, DistMult and ComplEx yield a higher validation MRR than ensembles of TransH or RotatE models (Figure 3a). When ensembling two different scoring functions, the best results are achieved by including DistMult or ComplEx (Figure 3c). Likewise, removing models using DistMult or ComplEx from an ensemble results in a substantial MRR degradation, while leaving out models with TransH or RotatE can even be beneficial (Figure 3d). A possible explanation for these observations can be found in the high training MRR achieved by DistMult and ComplEx models (Figure 2b), which produces good generalisation when these models' tendency to overfit is mitigated by the regularising effect of mean-ensembling.

Based on this evidence, individual models have been ranked by validation MRR and a diverse ensemble consisting of 85 models (the 25 best TransE, DistMult and ComplEx models, and the 5 best TransH and RotatE models; Figure 4a) was selected, achieving a validation MRR of 0.2922 and an MRR of 0.2562 on the test-challenge set. Changing the composition of this ensemble did not further improve validation MRR (Figure 4b).

7 Conclusions

We demonstrate the distributed training of large Knowledge Graph Embedding models on a Graphcore Bow Pod_{16} system. Enabled by the fast execution scheme of the distribution framework BESS, we show the substantial advantage of large ensembles of a diverse set of models over individual KGE models. With an MRR of 0.2562 on the test-challenge set, the solution laid out in this report is among the winning submissions to the WikiKG90Mv2 track of the Open Graph Benchmark Large-Scale Challenge at NeurIPS 2022.

Acknowledgements

We thank Luke Hudlass-Galley for his helpful comments on the manuscript. We are grateful for all the support received from our Graphcore colleagues.

References

Bollacker, K. D., Evans, C., Paritosh, P. K., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In J. T. Wang (Ed.), *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008* (pp. 1247–1250).

Bordes, A., Usunier, N., García-Durán, A., Weston, J., & Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013* (pp. 2787–2795).

Chao, L., Wang, T., & Chu, W. (2022). PIE: a parameter and inference efficient solution for large scale knowledge graph embedding reasoning. *CoRR*, *abs/2204.13957*.

Graphcore. (2022a). *Bow IPU processor.* `https://www.graphcore.ai/bow-processors`. (Online; accessed 18 November 2022)

Graphcore. (2022b). *Poplar graph framework software.* `https://www.graphcore.ai/products/poplar`. (Online; accessed 17 November 2022)

Hao, Y., Zhang, Y., Liu, K., He, S., Liu, Z., Wu, H., & Zhao, J. (2017). An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In R. Barzilay & M. Kan (Eds.), *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017* (pp. 221–231).

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., & Leskovec, J. (2021). OGB-LSC: A large-scale challenge for machine learning on graphs. In J. Vanschoren & S. Yeung (Eds.), *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021.*

Hu, W., Fey, M., Ren, H., Nakata, M., Dong, Y., & Leskovec, J. (2022). *OGB-LSC WikiKG90Mv2.* `https://ogb.stanford.edu/docs/lsc/wikikg90mv2/`. (Online; accessed 11 November 2022)

Jean, S., Cho, K., Memisevic, R., & Bengio, Y. (2015). On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1–10).

Kochsiek, A., & Gemulla, R. (2021). Parallel training of knowledge graph embedding models: A comparison of techniques. In *Proceedings of the VLDB Endowment* (Vol. 15, pp. 633–645).

Lacroix, T., Usunier, N., & Obozinski, G. (2018). Canonical tensor decomposition for knowledge base completion. In J. G. Dy & A. Krause (Eds.), *Proceedings of the 35th International Conference on Machine Learning, ICML 2018* (Vol. 80, pp. 2869–2878).

Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrstedt, L., Bose, A., & Peysakhovich, A. (2019). Pytorch-biggraph: A large scale graph embedding system. In A. Talwalkar, V. Smith, & M. Zaharia (Eds.), *Proceedings of Machine Learning and Systems 2019, MLSys 2019.*

Song, K., Tan, X., Qin, T., Lu, J., & Liu, T. (2020). MPNet: Masked and permuted pre-training for language understanding. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020.*

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Sun, Z., Deng, Z., Nie, J., & Tang, J. (2019). RotatE: Knowledge graph embedding by relational rotation in complex space. In *7th International Conference on Learning Representations, ICLR 2019.*

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., & Bouchard, G. (2016). Complex embeddings for simple link prediction. In M. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016* (Vol. 48, pp. 2071–2080).

Vrandecic, D., & Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Commun. ACM*, *57*(10), 78–85.

Wang, Z., Zhang, J., Feng, J., & Chen, Z. (2014). Knowledge graph embedding by translating on hyperplanes. In C. E. Brodley & P. Stone (Eds.), *Proceedings of the aaai conference on artificial intelligence* (pp. 1112–1119).

Yang, B., Yih, W., He, X., Gao, J., & Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In Y. Bengio & Y. LeCun (Eds.), *3rd International Conference on Learning Representations, ICLR 2015.*

Zhang, F., Yuan, N. J., Lian, D., Xie, X., & Ma, W. (2016). Collaborative knowledge base embedding for recommender systems. In B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, & R. Rastogi (Eds.), *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 353–362).

Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., ... Karypis, G. (2020). DGL-KE: Training knowledge graph embeddings at scale. In J. X. Huang et al. (Eds.), *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020* (pp. 739–748).