Compressing Transformer-Based Semantic Parsing Models using Compositional Code Embeddings

Prafull Prakash[1]*, **Saurabh Kumar Shashidhar**[1]*, **Wenlong Zhao**[1]*,
Subendhu Rongali[1], **Haidar Khan**[2], and **Michael Kayser**[2]
[1]University of Massachusetts Amherst
[2]Amazon Alexa
{prafullpraka, ssaurabhkuma, wenlongzhao, srongali}@cs.umass.edu
{khhaida, mikayser}@amazon.com

Abstract

The current state-of-the-art task-oriented semantic parsing models use BERT or RoBERTa as pretrained encoders; these models have huge memory footprints. This poses a challenge to their deployment for voice assistants such as Amazon Alexa and Google Assistant on edge devices with limited memory budgets. We propose to learn compositional code embeddings to greatly reduce the sizes of BERT-base and RoBERTa-base. We also apply the technique to DistilBERT, ALBERT-base, and ALBERT-large, three already compressed BERT variants which attain similar state-of-the-art performances on semantic parsing with much smaller model sizes. We observe 95.15% $\sim$ 98.46% embedding compression rates and 20.47% $\sim$ 34.22% encoder compression rates, while preserving >97.5% semantic parsing performances. We provide the recipe for training and analyze the trade-off between code embedding sizes and downstream performances.

1 Introduction

Conversational virtual assistants, such as Amazon Alexa, Google Home, and Apple Siri, have become increasingly popular in recent times. These systems can process queries from users and perform tasks such as playing music and finding locations. A core component in these systems is a task-oriented semantic parsing model that maps natural language expressions to structured representations containing intents and slots that describe the task to perform. For example, the expression *Can you play some songs by Coldplay?* may be converted to *Intent: PlaySong, Artist: Coldplay*, and the expression *Turn off the bedroom light* may be converted to *Intent: TurnOffLight, Device: bedroom*.

Task-oriented semantic parsing is traditionally approached as a joint intent classification and slot filling task. Kamath and Das (2018) provide a comprehensive survey of models proposed to solve this task. Researchers have developed semantic parsers based on Recurrent Neural Networks (Mesnil et al., 2013; Liu and Lane, 2016; Hakkani-Tür et al., 2016), Convolutional Neural Networks (Xu and Sarikaya, 2013; Kim, 2014), Recursive Neural Networks (Guo et al., 2014), Capsule Networks (Sabour et al., 2017; Zhang et al., 2019), and slot-gated attention-based models (Goo et al., 2018).

The current state-of-the-art models on SNIPS (Coucke et al., 2018), ATIS (Price, 1990), and Facebook TOP (Gupta et al., 2018) datasets are all based on BERT-style (Devlin et al., 2018; Liu et al., 2019) encoders and transformer architectures (Chen et al., 2019; Castellucci et al., 2019; Rongali et al., 2020). It is challenging to deploy these large models on edge devices and enable the voice assistants to operate locally instead of relying on central cloud services, due to the limited memory budgets on these devices. However, there has been a growing push towards the idea of TinyAI [1].

In this paper, we aim to build space-efficient task-oriented semantic parsing models that produce near state-of-the-art performances by compressing existing large models. We propose to learn compositional code embeddings to significantly compress BERT-base and RoBERTa-base encoders with little performance loss. We further use ALBERT-base/large (Lan et al., 2019) and DistilBERT (Sanh et al., 2019) to establish light baselines that achieve similar state-of-the-art performances, and apply the same code embedding technique. We show that our technique is complementary to the compression techniques used in ALBERT and DistilBERT. With all variants, we achieve 95.15% $\sim$ 98.46% embedding compression rates and 20.47% $\sim$ 34.22% encoder compression rates, with >97.5% semantic

*Equal contribution, alphabetical order

[1]https://www.technologyreview.com/technology/tiny-ai/

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 4711–4717

parsing performance preservation.

2 Related Compression Techniques

2.1 BERT Compression

Many techniques have been proposed to compress BERT (Devlin et al., 2018). Ganesh et al. (2020) provide a survey on these methods. Most existing methods focus on alternative architectures in transformer layers or learning strategies.

In our work, we use DistilBERT and ALBERT-base as light pretrained language model encoders for semantic parsing. DistilBERT (Sanh et al., 2019) uses distillation to pretrain a model that is 40% smaller and 60% faster than BERT-base, while retaining 97% of its downstream performances. ALBERT (Lan et al., 2019) factorizes the embedding and shares parameters among the transformer layers in BERT and results in better scalability than BERT. ALBERT-xxlarge outperforms BERT-large on GLUE (Wang et al., 2018), RACE (Lai et al., 2017), and SQUAD (Rajpurkar et al., 2016) while using less parameters.

We use compositional code learning (Shu and Nakayama, 2017) to compress the model embeddings, which contain a substantial amount of model parameters. Previously ALBERT uses factorization to compress the embeddings. We find more compression possible with code embeddings.

2.2 Embedding Compression

Varied techniques have been proposed to learn compressed versions of non-contextualized word embeddings, such as, Word2Vec (Mikolov et al., 2013) and GLoVE (Pennington et al., 2014). Subramanian et al. (2018) use denoising k-sparse autoencoders to achieve binary sparse intrepretable word embeddings. Chen et al. (2016) achieve sparsity by representing the embeddings of uncommon words using sparse linear common combination of common words. Lam (2018) achieve compression by quantization of the word embeddings by using 1-2 bits per parameter. Faruqui et al. (2015) use sparse coding in a dictionary learning setting to obtain sparse, non-negative word embeddings. Raunak (2017) achieve dense compression of word embeddings using PCA combined with a post-processing algorithm. Shu and Nakayama (2017) propose to represent word embeddings using compositional codes learnt directly in end-to-end fashion using neural networks. Essentially few common basis vectors are learnt and embeddings are reconstructed using their composition via a discrete code vector specific to each token embedding. This results in 98% compression rate in sentiment analysis and 94% - 99% in machine translation tasks without performance loss with LSTM based models. All the above techniques are applied to embeddings such as WordVec and Glove, or LSTM models.

We aim to learn space-efficient embeddings for transformer-based models. We focus on compositional code embeddings (Shu and Nakayama, 2017) since they maintain the vector dimensions, do not require special kernels for calculating in a sparse or quantized space, can be finetuned with transformer-based models end-to-end, and achieve extremely high compression rate. Chen et al. (2018) explores similar idea as Shu and Nakayama (2017) and experiment with more complex composition functions and guidances for training the discrete codes. Chen and Sun (2019) further show that end-to-end training from scratch of models with code embeddings is possible. Given various pretrained language models, we find that the method proposed by Shu and Nakayama (2017) is straightforward and perform well in our semantic parsing experiments.

3 Method

3.1 Compositional Code Embeddings

Shu and Nakayama (2017) apply additive quantization (Babenko and Lempitsky, 2014) to learn compositional code embeddings to reconstruct pretrained word embeddings such as GloVe (Pennington et al., 2014), or task-specific model embeddings such as those from an LSTM neural machine translation model. Compositional code embeddings E^C for vocabulary V consist of a set of M codebooks E^C_1, E^C_2, ..., E^C_M, each with K basis vectors of the same dimensionality D as the reference embeddings E, and a discrete code vector $(C^1_w, C^2_w, ..., C^M_w)$ for each token w in the vocabulary. The final embedding for w is composed by summing up the C^i_wth vector from the ith codebook as $E^C(C_w) = \sum_{i=1}^{M} E^C_i(C^i_w)$. Codebooks and discrete codes are jointly learned using the mean squared distance objective: $(C^*, E^{C*}) = \arg\min_{C,E^C} \frac{1}{|V|} \sum_{w \in V} ||E^C(C_w) - E(w)||^2$. For learning compositional codes, the Gumbel-softmax reparameterization trick (Jang et al., 2016; Maddison et al., 2016) is used for one-hot vectors corresponding to each discrete code.

Encoder	EncoderParam# / Size	EmbParam# / Size	SizeRatio	CCEmbSize	CCEncoderSize	EmbComp	EncoderComp
RoBERTa-base	125.29M / 477.94MB	38.60M / 147.25MB	30.81%	2.27MB	332.96MB	98.46%	30.33%
BERT-base-uncased	110.10M / 420.00MB	23.44M / 89.42MB	21.29%	1.97MB	332.55MB	97.80%	20.82%
DistilBERT-base-uncased	66.99M / 255.55MB	23.44M / 89.42MB	34.99%	1.97MB	168.10MB	97.80%	34.22%
ALBERT-large-v2	17.85M / 68.09MB	3.84M / 14.65MB	21.52%	0.71MB	54.15MB	95.15%	20.47%
ALBERT-base-v2	11.81M / 45.05MB	3.84M / 14.65MB	32.52%	0.71MB	31.11MB	95.15%	30.94%

Table 1: Model compression with compositional code ("cc") embeddings. The embedding layers are compressed by more than 95% with compositional code embeddings in all of the BERT variants.

3.2 Transformer-Based Models with Compositional Code Embeddings

In this work, we learn compositional code embeddings to reduce the size of the embeddings in pretrained contextualized language models. We extract the embedding tables from pretrained RoBERTa-base (Liu et al., 2019), BERT-base (Devlin et al., 2018), DistilBERT-base (Sanh et al., 2019), ALBERT-large-v2 and ALBERT-base-v2 (Lan et al., 2019) from the huggingface transformers library (Wolf et al., 2019) and follow the approach presented by Shu and Nakayama (2017) to learn the code embeddings. We then replace the embedding tables in the transformer models with the compositional code approximations and evaluate the compressed language models by finetuning on downstream tasks. When Shu and Nakayama (2017) feed compositional code embeddings into the LSTM neural machine translation model, they fix the embedding parameters and train the rest of the model from random initial values. In our experiments, we fix the discrete codes, initialize the transformer layers with those from the pretrained language models, initialize the task-specific output layers randomly, and finetune the codebook basis vectors with the rest of the non-discrete parameters.

3.3 Size Advantage of Compositional Code Embeddings

An embedding matrix $E \in \mathbb{R}^{|V| \times D}$ stored as 32-bit float point numbers, where $|V|$ is the vocabulary size and D is the embedding dimension, requires $32|V|D$ bits. Its compositional code reconstruction requires $32MKD$ bits for MK basis vectors, and $M \log_2 K$ bits for codes of each of $|V|$ tokens. Since each discrete code takes an integer value in $[1, K]$, it can be represented using $\log_2 K$ bits.

Table 1 illustrates the size advantage of compositional code embeddings for various pretrained transformer models (Wolf et al., 2019) used in our experiments. While the technique focuses on compressing the embedding table, it is compatible with other compression techniques for transformer models, including parameter sharing among transformer layers and embedding factorization used in ALBERT and distillation for learning DistilBERT. In our experiments, we apply the code learning technique to compress embeddings in five pretrained BERT variants by 95.15% $\sim$ 98.46% to build competitive but significantly lighter semantic parsing models.

Dataset	Train	Valid	Test	#Intent	#Slot
ATIS	4,478	500	893	26	83
SNIPS	13,084	700	700	7	39
Facebook TOP	31,279	4,462	9,042	25	36

Table 2: Statistics for semantic parsing datasets.

4 Datasets

Following Rongali et al. (2020), we evaluate our models on SNIPS (Coucke et al., 2018), Airline Travel Information System (ATIS) (Price, 1990), and Facebook TOP (Gupta et al., 2018) datasets for task-oriented semantic parsing (Table 2). For SNIPS and ATIS, we use the same train/validation/test split as Goo et al. (2018).

5 Experiments and Analyses

For transformer model training, we base our implementation on the huggingface transformers library v2.6.0 (Wolf et al., 2019). We use the AdamW optimizer (Loshchilov and Hutter, 2017) with 10% warmup steps and linear learning rate decay to 0. Forr code embedding learning, we base our implementation on that of Shu and Nakayama (2017). By default we learn code embeddings with 32 codebooks and 16 basis vectors per codebook. Unless otherwise specified, hyperparameters are found according to validation performances from one random run. We conduct our experiments on a mixture of Tesla M40, TITAN X, 1080 Ti, and 2080 Ti GPUs. We use exact match (EM) and intent accuracy as evaluation metrics. Exact match requires correct predictions for all intents and slots in a query, and is our primary metric.

Model					EM	Intent
Joint BiRNN (Hakkani-Tür et al., 2016)					73.2	96.9
Attention BiRNN (Liu and Lane, 2016)					74.1	96.7
Slot Gated Full Attention (Goo et al., 2018)					75.5	97.0
CapsuleNLU (Zhang et al., 2019)					80.9	97.3
BERT-Seq2Seq-Ptr (Rongali et al., 2020)					86.3	98.3
RoBERTa-Seq2Seq-Ptr (Rongali et al., 2020)					87.1	98.0
BERT-Joint (Castellucci et al., 2019)					91.6	**99.0**
Joint BERT (Chen et al., 2019)					**92.8**	98.6
Ours	**epo**	**lr**	**wd**	**EM-v**	**EM**	**Intent**
ALBERT-base		5e-5	0.05	90.71	**91.29**	98.86
ALBERT-base_cc	1100	5e-5	0.01	90.00	89.14	98.14
ALBERT-large		3e-5	0.05	91.29	**92.43**	98.14
ALBERT-large_cc	1100	2e-5	0.05	91.14	**92.43**	98.71
DistilBERT-base		3e-5	0.05	90.29	91.14	98.57
DistilBERT-base_cc	900	6e-5	0.01	90.14	**91.24**	98.43
BERT-base		3e-5	0.05	92.14	**92.29**	99.14
BERT-base_cc	900	6e-5	0.05	91.29	90.71	98.71

Table 3: Results on SNIPS. "cc" indicate models with code embeddings. "epo" is the epoch number for offline code embedding learning. "lr" and "wd" are the peak learning rate and weight decay for whole model finetuning. "EM-v", "EM", "Intent" indicate validation exact match, test exact match, and test intent accuracy.

Model					EM	Intent
Joint-BiRNN (Hakkani-Tür et al., 2016)					80.7	92.6
Attention-BiRNN (Liu and Lane, 2016)					78.9	91.1
Slot-Gated (Goo et al., 2018)					82.2	93.6
CapsuleNLU (Zhang et al., 2019)					83.4	95.0
BERT-Seq2Seq-Ptr (Rongali et al., 2020)					86.4	97.4
RoBERTa-Seq2Seq-Ptr (Rongali et al., 2020)					87.1	97.4
BERT-Joint (Castellucci et al., 2019)					**88.2**	**97.8**
Joint-BERT (Chen et al., 2019)					**88.2**	97.5
Ours	**epo**	**lr**	**wd**	**EM-v**	**EM**	**Intent**
ALBERT-base		5e-5	0.05	93.4	86.90	97.42
ALBERT-base_cc	900	6e-5	0.1	94.2	**87.23**	96.75
ALBERT-large		5e-5	0.05	93.8	**88.02**	97.54
ALBERT-large_cc	1100	5e-5	0.1	94.0	87.91	97.54
DistilBERT-base		4e-5	0.05	93.6	**88.13**	97.42
DistilBERT-base_cc	1100	6e-5	0.05	93.2	87.12	97.54
BERT-base		4e-5	0.01	93.4	**88.13**	97.54
BERT-base_cc	700	6e-5	0.1	93.0	87.35	97.20

Table 4: Results on ATIS. Refer to the caption of Table 3 for abbreviation explanations.

5.1 SNIPS and ATIS

We implement a joint sequence-level and token-level classification layer for pretrained transformer models. The intent probabilities are predicted as $y^i = \text{softmax}(W^i h_0 + b^i)$, where h_0 is the hidden state of the [CLS] token. The slot probabilities for each token j are predicted as $y^s_j = \text{softmax}(W^s h_j + b^s)$. We use the cross entropy loss to maximize $p(y^i|x) \prod p(y^s_j|x)$ where j is the first piece-wise token for each word in the query. We learn code embeddings for {500, 700, 900, 1100, 1300} epochs. We train transformer models with original and code embeddings all for 40 epochs with batch size 16 and sequence length 128. Uncased BERT and DistilBERT perform better than the cased versions. We experiment with peak learning rate {2e-5, 3e-5, ..., 6e-5} and weight decay {0.01, 0.05, 0.1}. As shown in Table 3 and 4, we use different transformer encoders to establish strong baselines which achieve EM values that are within 1.5% of the state-of-the-art.

On both datasets, models based on our compressed ALBERT-large-v2 encoder (54MB) perserves >99.6% EM of the previous state-of-the-art model (Chen et al., 2019) which uses a BERT encoder (420MB). In all settings, our compressed encoders perserve >97.5% EM of the uncompressed counterparts under the same training settings. We show that our technique is effective on a variety of pretrained transformer encoders.

Model	EM-v	EM	Intent
RNNG (Gupta et al., 2018)		78.51	-
Shift Reduce (SR) Parser		80.86	-
SR with ELMo embeddings		83.93	-
SR ensemble + ELMo + SVMRank		**87.25**	-
BERT-Seq2Seq-Ptr (Rongali et al., 2020)		83.13	97.91
RoBERTa-Seq2Seq-Ptr (Rongali et al., 2020)		86.67	**98.13**
Ours	**EM-v**	**EM**	**Intent**
ALBERT-Seq2Seq-Ptr	84.56	**85.41**	98.47
ALBERT-Seq2Seq-Ptr_cc	83.48	84.42	98.05
DistilBERT-Seq2Seq-Ptr	84.25	**85.12**	98.50
DistilBERT-Seq2Seq-Ptr_cc	82.76	83.42	98.09
BERT-Seq2Seq-Ptr	83.83	**85.01**	98.59
BERT-Seq2Seq-Ptr_cc	82.36	83.34	98.25
RoBERTa-Seq2Seq-Ptr	85.00	**85.67**	98.59
RoBERTa-Seq2Seq-Ptr_cc	83.51	83.78	98.17

Table 5: Results on Facebook TOP. The SR models are by Einolghozati et al. (2019). Refer to the caption of Table 3 for abbreviation explanations.

5.2 Facebook TOP

Table 5 presents results on Facebook TOP. We follow Rongali et al. (2020) and experiment with Seq2Seq models. We use different pretrained BERT-variants as the encoder, transformer decoder layers with $d_{model} = 768$ (Vaswani et al., 2017), and a pointer generator network (Vinyals et al., 2015) which uses scaled dot-product attention to score tokens. The model is trained using the cross-entropy loss with label smoothing of 0.1. For simplicity, we always train code embeddings for 900 epochs offline. Learning rate 2e-5 and weight decay 0.01 are used for transformer training. BERT and DistilBERT are cased in these experiments. During inference, we employ beam decoding with width 5. Our greatly compressed models present 98∼99% performances of the original models.

Epoch	MeanEucDist	NN-cos	NN-Euc	SNIPS	ATIS	TOP
100	0.3677±0.25%	0.66±1.90%	0.65±2.00%	79.29	82.31	78.09
200	0.3254±0.08%	2.20±0.69%	2.30±0.84%	85.43	84.99	81.59
300	0.3023±0.09%	3.66±0.92%	3.96±0.55%	86.86	86.11	83.17
400	0.2841±0.23%	4.84±0.58%	5.26±0.83%	**89.71**	87.01	83.45
500	0.2685±0.26%	5.72±0.48%	6.21±0.78%	87.71	87.23	83.82
600	0.2573±0.12%	6.20±0.39%	6.72±0.18%	88.14	85.69	83.41
700	0.2499±0.20%	6.42±0.49%	6.94±0.33%	88.00	**87.35**	84.27
800	0.2444±0.07%	6.54±0.39%	7.07±0.15%	88.57	86.90	84.09
900	0.2407±0.10%	6.62±0.31%	7.14±0.14%	88.57	86.56	**84.42**
1000	0.2380±0.07%	6.65±0.39%	7.16±0.10%	89.14	87.12	83.86

Table 6: Analyses for the code embedding learning process (M=32, K=16). MeanEucDist, NN-cos, and NN-Euc are averaged across 5 runs. "SNIPS", "ATIS", and "TOP" are the test exact match achieved on the three datasets.

5.3 Analysis for Code Convergence

We study the relationship among a few variables during code learning for the embeddings from pretrained ALBERT-base (Table 6). During the first 1000 epochs, the mean Euclidean distance between the original and reconstructed embeddings decrease with a decreasing rate. The average number of shared top-20 nearest neighbours according to cosine similarity and Euclidean distances between the two embeddings increase with a decreasing rate. We apply code embeddings trained for different numbers of epochs to ALBERT-base-v2 and finetune on semantic parsing. On SNIPS and ATIS, we find the best validation setting among learning rate {2,3,4,5,6}e-5 and weight decay {0.01, 0.05, 0.01}. We observe that the test exact match plateaus for code embeddings trained for more than 400 epochs. On Facebook TOP, we use learning rate 2e-5 and weight decay 0.01, and observe the similar trend.

5.4 Effects of M and K

We use embeddings from pretrained ALBERT-base-v2 as reference to learn code embeddings with M in {8, 16, 32, 64} and K in {16, 32, 64}. As shown in Table 7, after 700 epochs, the MSE loss for embeddings with *larger* M and K converges to *smaller* values in general. With M=64, more epochs are needed for convergence to smaller MSE losses compared to those from smaller M. We apply the embeddings to ALBERT-base-v2 and finetune on SNIPS. In general, larger M yields better performances. Effects of K are less clear when M is large.

M	K	epo	MSE	EM
8	16	700	0.3155±0.05%	85.43
8	32	700	0.3032±0.04%	87.43
8	64	700	0.2944±0.04%	87.43
16	16	700	0.2855±0.05%	88.57
16	32	700	0.2727±0.09%	88.00
16	32	700	0.2669±0.08%	88.14
32	16	700	0.2499±0.20%	89.00
32	32	700	0.2421±0.20%	89.14
32	64	700	0.2396±0.27%	88.29
64	16	700	0.2543±0.47%	88.29
64	16	1000	0.2256±1.06%	89.71
64	32	700	0.2557±0.37%	89.86
64	32	1000	0.2159±0.43%	89.71

Table 7: Effects of M and K. Mean squared errors (MSE) are averaged over 5 runs. Best validation exact match (EM) is presented for compressed transformer models trained with 0.05 weight decay and {3,4,5,6,7}e-5 peak learning rates on SNIPS.

6 Conclusion

Current state-of-the-art task-oriented semantic parsing models are based on pretrained RoBERTa-base (478MB) or BERT-base (420MB). We apply DistilBERT (256MB), ALBERT-large (68MB), and ALBERT-base (45MB), and observe near state-of-the-art performances. We learn compositional code embeddings to compress the model embeddings by 95.15% $\sim$ 98.46%, the pretrained encoders by 20.47% $\sim$ 34.22%, and observe 97.5% performance preservation on SNIPS, ATIS, and Facebook TOP. Our compressed ALBERT-large is 54MB and can achieve 99.6% performances of the previous state-of-the-art models on SNIPS and ATIS. Our technique has potential to be applied to more tasks including machine translation in the future.

Acknowledgement

This project is part of the data science industry mentorship program initiated by Andrew McCallum at University of Massachusetts Amherst. We thank the teaching assistants Rajarshi Das and Xiang Lorraine Li for helpful discussion and the instructor Andrew McCallum for valuable feedback. Experiments in this project are conducted on the Gypsum cluster at UMassAmherst. The cluster is purchased with funds from the Massachusetts Technology Collaborative.

References

Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 931–938.

Giuseppe Castellucci, Valentina Bellomaria, Andrea Favalli, and Raniero Romagnoli. 2019. Multilingual intent detection and slot filling in a joint bert-based model. *arXiv preprint arXiv:1907.02884*.

Qian Chen, Zhu Zhuo, and Wen Wang. 2019. Bert for joint intent classification and slot filling.

Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. *arXiv preprint arXiv:1806.09464*.

Ting Chen and Yizhou Sun. 2019. Differentiable product quantization for end-to-end embedding compression. *arXiv preprint arXiv:1908.09756*.

Yunchuan Chen, Lili Mou, Yan Xu, Ge Li, and Zhi Jin. 2016. Compressing neural language models by sparse word representations. *arXiv preprint arXiv:1610.03950*.

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, et al. 2018. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Arash Einolghozati, Panupong Pasupat, Sonal Gupta, Rushin Shah, Mrinal Mohit, Mike Lewis, and Luke Zettlemoyer. 2019. Improving semantic parsing for task oriented dialog. *arXiv preprint arXiv:1902.06000*.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah Smith. 2015. Sparse overcomplete word vector representations. *arXiv preprint arXiv:1506.02004*.

Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Deming Chen, Marianne Winslett, Hassan Sajjad, and Preslav Nakov. 2020. Compressing large-scale transformer-based models: A case study on bert. *arXiv preprint arXiv:2002.11985*.

Chih-Wen Goo, Guang Gao, Yun-Kai Hsu, Chih-Li Huo, Tsung-Chieh Chen, Keng-Wei Hsu, and Yun-Nung Chen. 2018. Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 753–757.

Daniel Guo, Gokhan Tur, Wen-tau Yih, and Geoffrey Zweig. 2014. Joint semantic utterance classification and slot filling with recursive neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, pages 554–559. IEEE.

Sonal Gupta, Rushin Shah, Mrinal Mohit, Anuj Kumar, and Mike Lewis. 2018. Semantic parsing for task oriented dialog using hierarchical representations. *arXiv preprint arXiv:1810.07942*.

Dilek Hakkani-Tür, Gökhan Tür, Asli Celikyilmaz, Yun-Nung Chen, Jianfeng Gao, Li Deng, and Ye-Yi Wang. 2016. Multi-domain joint semantic frame parsing using bi-directional rnn-lstm. In *Interspeech*, pages 715–719.

Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Aishwarya Kamath and Rajarshi Das. 2018. A survey on semantic parsing. *arXiv preprint arXiv:1812.00978*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. 2017. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*.

Maximilian Lam. 2018. Word2bits-quantized word vectors. *arXiv preprint arXiv:1803.05651*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Bing Liu and Ian Lane. 2016. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv preprint arXiv:1609.01454*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. 2016. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*.

Grégoire Mesnil, Xiaodong He, Li Deng, and Yoshua Bengio. 2013. Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding. In *Interspeech*, pages 3771–3775.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Patti Price. 1990. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*.

Vikas Raunak. 2017. Simple and effective dimensionality reduction for word embeddings. *arXiv preprint arXiv:1708.03629*.

Subendhu Rongali, Luca Soldaini, Emilio Monti, and Wael Hamza. 2020. Don't parse, generate! a sequence to sequence architecture for task-oriented semantic parsing. *arXiv preprint arXiv:2001.11458*.

Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. 2017. Dynamic routing between capsules. *CoRR*, abs/1710.09829.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Raphael Shu and Hideki Nakayama. 2017. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*.

Anant Subramanian, Danish Pruthi, Harsh Jhamtani, Taylor Berg-Kirkpatrick, and Eduard Hovy. 2018. Spine: Sparse interpretable neural embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in neural information processing systems*, pages 2692–2700.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *CoRR*, abs/1804.07461.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Puyang Xu and Ruhi Sarikaya. 2013. Convolutional neural network based triangular crf for joint intent detection and slot filling. In *2013 ieee workshop on automatic speech recognition and understanding*, pages 78–83. IEEE.

Chenwei Zhang, Yaliang Li, Nan Du, Wei Fan, and Philip Yu. 2019. Joint slot filling and intent detection via capsule neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy. Association for Computational Linguistics.