Zero-Shot Learning across Heterogeneous Overlapping Domains

Anjishnu Kumar[1], *Pavankumar Reddy Muddireddy*[2], *Markus Dreyer*[1], *Björn Hoffmeister*[1]

[1]Amazon Inc, USA [2]University of Illinois at Urbana-Champaign, USA

anjikum@amazon.com, muddire2@illinois.edu, mddreyer@amazon.com, bjornh@amazon.com

Abstract

We present a zero-shot learning approach for text classification, predicting which natural language understanding domain can handle a given utterance. Our approach can predict domains at runtime that did not exist at training time. We achieve this extensibility by learning to project utterances and domains into the same embedding space while generating each domain-specific embedding from a set of attributes that characterize the domain. Our model is a neural network trained via ranking loss. We evaluate the performance of this zero-shot approach on a subset of a virtual assistant's third-party domains and show the effectiveness of the technique on new domains not observed during training. We compare to generative baselines and show that our approach requires less storage and performs better on new domains.

1. Introduction

Traditionally, virtual assistants like Alexa, Cortana and the Google Assistant have had a small and relatively fixed number of domains, which are groupings of mutually related user intents, and predicting the right domain for a given utterance could be treated as a multi-class classification problem [15]. However, with the advent of new frameworks like the Alexa Skills Kit, the Cortana Skills Kit, and Actions on Google, the number of domains has started exponentially, following the patterns followed by applications on smartphones, they are increasingly written by non-experts and may have completely heterogenous and/or overlapping output label spaces. A domain classifier model that has to be retrained from scratch for every new domain is infeasible. While the domain prediction model in this kind of setting can be retrained with new domains at regular intervals, the system should also be able to handle new domains out of the box in the interim period between the training regimes.

We achieve this continuous extensibility to new domains by first learning a function that can project any domain into a dense vector. That is, we learn a function that can generate a domain embedding for any domain. As input, this function takes *attributes* of the domain, which characterize it, e.g., in terms of the sample utterances that it defines, its developer-specified category, or other metadata. This function, which generates domain embeddings from domain attributes, is learned jointly with another function that can generate an utterance embedding for any incoming utterance. Our joint training objective encourages these two functions to use the same embedding space so that, for example, utterances that ask for a cab are projected into the same space as domains that can call cabs. When training is completed, we apply these functions to any incoming utterance and all domains – including domains that we have not observed during training, and list the domains whose embeddings are most similar to the utterance embedding. With domain attributes, we try to offset the effect of not having observed a new domain during training by defining attributes that "explain" that domain.

The described model is a neural joint attribute learning framework for performing zero-shot classification into the new domains along with the existing domains. Unlike the built-in domains in virtual assistants, third-party domains are often overlapping, such that multiple domains apply to any given utterance (see Table 1). We predict a top-k list of suitable domains per utterance (i.e., we create a domains shortlister), which could then be passed to business logic that considers personalized information like user preferences or past interactions.

Utterance	**Domain(s)**
Get me a car	Uber, Lyft, ...
Find me a flight from Seattle to Boston	Kayak, Skyscanner, ...
Open my garage door	Garageio, Nexx Garage, ...
How long did I sleep last night	Fitbit, ...
When is high tide in San Diego on Saturday	TideReport, TidePooler, ...

Table 1: *Typical utterances and domains that can handle them.*

2. Zero-Shot Learning

In zero-shot learning, we are presented with training data for a set of output classes, but have to predict on unseen classes while testing; there is no training data for the unseen classes. This paper deals with the case where novel classes (i.e., domains) are added after our model has been trained, and we are constrained to not retrain to incorporate these new classes. This is a realistic problem setting as large model training is a computationally expensive process, and third-party developers can continuously add new domains.

2.1. Proposed Zero-Shot Architecture

Standard classifiers cannot be applied to zero-shot settings because they learn unique parameters per training class $y \in \mathcal{Y}^{\text{train}}$. So they cannot generalize to predict new classes at test time that have not been observed during training, i.e., when $\mathcal{Y}^{\text{train}} \subsetneq \mathcal{Y}^{\text{test}}$. To score an input/output pair (x, y), for example, standard feedforward neural networks use a scoring function [1] that has one parameter vector per training class y,

$$s(x, y; \boldsymbol{\theta}, \mathbf{f}_x) = \mathbf{h}_x(x; \boldsymbol{\theta}_x, \mathbf{f}_x) \cdot \boldsymbol{\theta}_y^T \qquad (1)$$

where $\mathbf{f}_x$ is a function that extracts *input attributes* (i.e., features) of the input x; $\boldsymbol{\theta}_x$ are the parameters of the neural network excluding the final layer; the hidden layer $\mathbf{h}_x$ is a dense embedding representation of the input based on its attributes $\mathbf{f}_x(x)$; and $\boldsymbol{\theta}_y$ are the final layer parameters corresponding to class y. Note that the function is linear in the class parameters $\boldsymbol{\theta}_y$. Such models are not applicable to zero-shot learning because only the parameters $\boldsymbol{\theta}_y$ corresponding to $y \in \mathcal{Y}^{\text{train}}$ are trained while no training has been performed on parameters corresponding to any unseen classes $y \in \mathcal{Y}^{\text{test}} \setminus \mathcal{Y}^{\text{train}}$. While such models can, at test time, process *inputs* that have never been observed during training – because inputs are represented by their input attributes $\mathbf{f}_x(x)$ – they cannot predict outputs y that have never

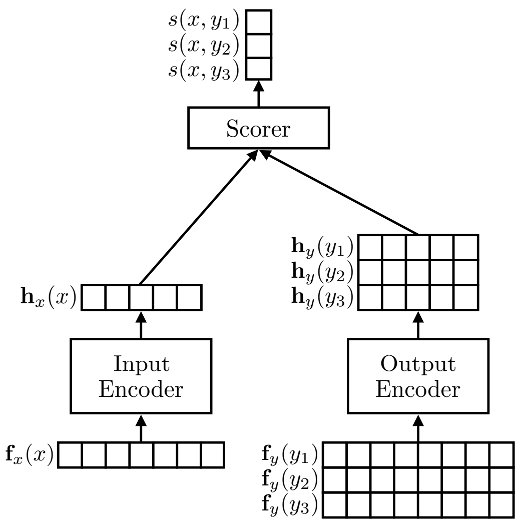

Figure 1: *Visualization: zero-shot model for input x and an example case of three output classes. Vectors $\mathbf{f}_x$ and $\mathbf{f}_y$ are attributes; $\mathbf{h}_x$ and $\mathbf{h}_y$ are dense embeddings. At test time, new classes can be scored along with classes observed during training.*

been observed during training.

To address this problem of unobserved output classes at test time, we introduce *class attributes* $\mathbf{f}_y(y)$, analogous to *input attributes* $\mathbf{f}_x(x)$, and based on these, we encode each output class into a dense class embedding $\mathbf{h}_y$, analogous to the dense input embedding $\mathbf{h}_x$. The hidden embeddings of the input and the outputs are trained to be in the same embedding space (Sec. 2.5). For any pair (x, y), a score can be computed based on the dense embeddings $\mathbf{h}_x(x)$ and $\mathbf{h}_y(y)$;[1] see Fig. 1 for a visualization of the overall model. The functions $\mathbf{f}_x$ and $\mathbf{f}_y$ are defined before training starts, such that each input x and each output class y, no matter if they have been observed during training or not, can be expressed with respect to their attributes, and so hidden input and class embeddings can be computed and scored for any candidate pair (x, y). Our scoring function becomes

$$s(x, y; \boldsymbol{\theta}, \mathbf{f}_x, \mathbf{f}_y) = \mathbf{h}_x(x; \boldsymbol{\theta}_x, \mathbf{f}_x) \cdot \mathbf{h}_y(y; \boldsymbol{\theta}_y, \mathbf{f}_y)^T \qquad (2)$$

where $\mathbf{h}_y(y; \boldsymbol{\theta}_y, \mathbf{f}_y)$ is the embedding of class y, based on the class attributes $\mathbf{f}_y(y)$, and $\boldsymbol{\theta}_y$ is a set of parameters shared for all classes. With this architecture, instead of learning parameters for individual classes, we learn parameters to project attributes into an embedding space, for inputs x as well as classes y, resulting in the capability to generalize not only to new inputs, but also to new classes not observed in training.

At a high level, the framework consists of three components: an *input encoder*, an *output encoder* and a *discriminator* or *scorer* module. The input encoder computes an input embedding given the input attributes, the output encoder does the same for each output class given its class attributes. The scorer tries to produce a high score for a valid input and class combination while producing low scores for invalid ones, which in turn encourages the input and output encoders to project into the same embedding space. Each of these modules is fully differentiable, and so the system can be trained end to end using backpropagation.

[1]For brevity, we write $\mathbf{h}_x(x)$ for $\mathbf{h}_x(x; \boldsymbol{\theta}_x, \mathbf{f}_x)$, etc.

2.2. Input Encoder

The input encoder accepts the attributes of an input utterance, $\mathbf{f}_x(x)$, and computes a dense embedding $\mathbf{h}_x(x)$. The input attributes can include all utterance-specific contextual features like the time of day, the text of the previous utterance, speech recognition accuracy, etc. For the purpose of this paper, we limit ourselves to adopting a feature representation for an utterance using only its words. It is a simplified version of the architecture adopted by [4]. In order to share parameters between similar words and to generalize to unseen words during test, we use 300-dimensional pre-trained word embeddings [14] for initializing the lookup layer. This is followed by a mean pooling layer followed by an affine layer with a $(\tanh)$ nonlinear activation function. Since the utterances are usually short, this architecture produced competitive results compared to an LSTM-based architecture. Recent work on applying neural network to paraphrasing show similar architectures outperforms LSTM-based architectures [21] for short utterances.

2.3. Output Encoder

The output encoder accepts the attributes of a candidate output class, $\mathbf{f}_y(y)$, and computes a dense embedding $\mathbf{h}_y(y)$.[2] Similar to the input encoder, the output encoder is a 256-dimensional dense layer. In our case, each class y is a natural language understanding (NLU) domain. For each domain y, we extract the following attributes $\mathbf{f}_y(y)$:

1. **Category metadata:** Developer-provided metadata such as domain category (e.g. 'News', 'Travel' or 'Home Automation').
2. **Mean-pooled word embeddings:** For each domain, we obtain a number of sample utterances (e.g., *get me a car to the station*) from a developer-provided domain grammar. We average the pre-trained word embeddings for each word in each sample utterance, then average these over all sample utterances; these are not updated during training.
3. **Gazetteer attributes:** We have a number of in-house gazetteers, e.g., artists, locations, organizations. Each domain is characterized by the gazetteers that its sample utterances match. Gazetteer-firing patterns are noisy, and some gazetteers are badly constructed, so instead of using raw matches against the gazetteers as feature values, we normalize them by applying TF-IDF.

2.4. Scorer

The scorer takes as input the embedding of the input and the embeddings of the outputs and produces similarity scores of the input and the outputs. We simply define the scorer as a vector dot product, but it can alternatively be defined as cosine distance, Euclidean distance or as a trainable neural network in itself, jointly trained as part of the larger network.

2.5. Learning and Inference

We now describe how learning and inference work using our model in a zero-shot setting. Let $\mathcal{D}^{\text{train}} = \{(x_i, y_i)\}_{i=1}^N$ denote the available training data with $y_i \in \mathcal{Y}^{\text{train}} \, \forall \, i$. With the scoring function as defined in Eq. 2, we could define a probability dis-

[2]It can also compute the embeddings for *all* candidate classes in parallel, producing a matrix of embeddings, one per class, as in Fig. 1.

tribution over the training classes $\mathcal{Y}^{\text{train}}$ using a softmax layer similar to a maximum entropy model,

$$p(y|x) = \frac{\exp\, s(x, y)}{\sum_{y' \in \mathcal{Y}^{\text{train}}} \exp\, s(x, y')} \tag{3}$$

We could then optimize the parameters $\boldsymbol{\theta}_x$ and $\boldsymbol{\theta}_y$ by minimizing a cross-entropy loss over the training classes $\mathcal{Y}^{\text{train}}$. However, since the test classes $\mathcal{Y}^{\text{test}}$ are different from $\mathcal{Y}^{\text{train}}$, optimizing over the distribution defined by Eq. 3, i.e., normalizing over $\mathcal{Y}^{\text{train}}$ while testing on $\mathcal{Y}^{\text{test}}$, is not well motivated.

Instead, we attained empirically stronger performance by using an SVM-like margin-based objective function, popular in the information retrieval literature [13]. We minimize as follows,

$$\min_{\boldsymbol{\theta}_x, \boldsymbol{\theta}_y} \sum_{i=1}^{N} [\, \max_{y \neq y_i} \left(s(x_i, y) + \gamma - s(x_i, y_i) \right) \,]_+ \tag{4}$$

where $[x]_+$ is the hinge function, which is equal to x when $x > 0$ else 0. This objective function tries to maximize the margin between the correct class and all the other classes.

During training, we start by sampling a random negative class label and maximizing the margin loss between that and the oracle true sample. After each epoch of training, we perform an inference step over the list of training classes and sample negative samples from that posterior distribution. This results in an implicit *curriculum learning* setup [2]. At the start of training, the model chooses random output classes, but as training progresses, the model starts choosing the hardest, most confusable cases as negative samples. Consistent with prior work [2], we find that this training strategy significantly speeds up convergence in training compared to purely random sampling, though sampling from the normalized output distribution adds a fixed time cost. Furthermore, maximizing the margin with the 'best' incorrect class implies that the margin with other incorrect classes is maximized as well.

Since the input feature representation is a feed-forward neural network in itself, we can optimize this objective in an online fashion using any gradient descent method like stochastic gradient descent (SGD). We train the model end to end using Dropout [20] on both encoder networks with a dropout rate of 0.2.

Denote the margin loss term $\max_{y \neq y_i} \left(s(x_i, y) + \gamma - s(x_i, y_i) \right)$ by ζ_i, i.e., the slack for instance i. We minimize this slack across all instances. For a single training sample (x_i, y_i), the sub-gradient with respect to any parameter is

$$\frac{\partial \zeta_i}{\partial \boldsymbol{\theta}} = \left(\frac{\partial s(x_i, \hat{y}_i)}{\partial \boldsymbol{\theta}} - \frac{\partial s(x_i, y_i)}{\partial \boldsymbol{\theta}} \right) \times [\![\zeta_i > 0]\!] \tag{5}$$

where $\hat{y}_i = \operatorname{argmax}_{y \neq y_i} s(x_i, y)$, i.e., the highest-scoring incorrect prediction under the current model, and y_i denotes the ground truth. $[\![\pi]\!] = 1$ if π is true, 0 otherwise. So, the sub-gradient is 0 when the slack, $\zeta_i \leq 0$, i.e., if $s(x_i, y_i) \geq s(x_i, \hat{y}_i) + \gamma$ in which case the margin condition is already satisfied. This leads to smaller updates as training proceeds and can be successfully exploited if incorporated into the implementation.

To compute the partial gradients during training on a sample (x_i, y_i), we compute the input embedding $\mathbf{h}_x(x_i)$ and all output class embeddings $\mathbf{h}_y(y)$ and resulting scores $s(x_i, y)\ \forall\ y \in \mathcal{Y}^{\text{train}}$ using Eq. 2 and determine $\hat{y}_i$. We then use Eq. 5 to compute the gradients for updating the parameters of the scoring function.

During testing, we are given a set of examples $\mathcal{D}^{\text{test}} = \{(x_i, y_i)\}_{i=1}^{N'}$ where $y_i \in \mathcal{Y}^{\text{test}}\, \forall\, i$. Provided we can compute $\mathbf{f}_y(y)\, \forall\, y \in \mathcal{Y}^{\text{test}}$, we can follow the process described previously to compute the score for all classes $\mathcal{Y}^{\text{test}}$ and predict the best class as $\operatorname{argmax}_{y \in \mathcal{Y}^{\text{test}}} s(x_i, y)$.

3. Related Work

The idea of using distributed representations for output classes is explored by [6]. They use error-correcting codes for classes resulting in compact representation and better generalization. The individual binary indicators in these error-correcting codes are generated randomly and does not carry any semantic meaning. There has been interest in using attributes to represent classes to successfully exploit class similarities and meta-data available for many problems in the computer vision community ([8], [7], [12], [16]). Many of the NLP problems can similarly be cast as attribute learning problems for better generalization and extending to novel classes. Some of the recent works that explore that aspect include [5], [3]. In structured prediction [19], the number of outputs can be exponential, and complete outputs for test instances have typically not been observed during training; however they are composed of atomic output units that have been observed. Meta-learning techniques such as those explored by [17] also open up an intriguing avenue for research on how to train networks to learning-to-learn from a few examples. Most of the works outlined above, however, do not consider a case in which the output label space contains both the training labels and test labels.

4. Experiments

4.1. Baselines

In addition to the zero-shot model, we also train a baseline model using k-nearest neighbors (k-NN) on domain embeddings in a procedure similar to [3]. Another baseline is a generative approach by decomposing the problem as $P(\text{domain} \mid \text{utterance}) \propto P(\text{utterance} \mid \text{domain}) \times P(\text{domain})$. This decomposition implies that we build independent models $P(\text{utterance} \mid \text{domain})$ for every domain and independently calculate domain priors. Here is a list of our baseline methods:

1. **Naïve Bayes (Unigram):** $P(\text{utterance} \mid \text{domain})$ is modeled with Naïve Bayes model with features being word unigrams in the utterance.
2. **Naïve Bayes (Unigram + Bigram):** Same as above but word bigrams are added as features in addition to unigrams.
3. **Language Model:** A trigram language model is used per domain to model $P(\text{utterance} \mid \text{domain})$. Kneser-Ney smoothing for n-gram backoff has been applied.
4. **Embeddings k-NN:** k-NN using intent embeddings from a classifier trained on data excluding the zero-shot partition.

We choose a vocabulary size of 10,000 unique words. Words not included in the vocabulary are mapped to a special rare word token, so their counts are shared per domain model.

Each domain consists of multiple intents, e.g., a *MyTaxi* domain may contain a *MyTaxi.GetTaxiIntent* and a *MyTaxi.CancelTaxiIntent*. Intents can be seen as fine-grained domains themselves; they are more homogeneous and therefore easier to model. We found that models perform better if they are trained to predict intents, which are then mapped back to their corresponding domains for evaluation. We use this technique for all models.

4.2. Experimental Setup

For third-party domains, sample utterances are generated from the domain grammars provided by the developers. Sample utterances cover the utterances that the developer expects the users of the domain would use to interact with it. For the discriminative model, we further use generated data from other domains during training to better learn feature-attribute association weights, although we need not test on these domains. For this purpose, data from 1,574 domains have been used while training the joint model. Although we have usage data for thousands of domains available, to show the promise of the joint attribute learning framework proposed for zero-shot learning in this paper, we restrict ourselves to testing on 32 third-party domains, which is a small subset of the domains for which we have annotated test data. Several of these test domains mutually overlap. We partition the test set into three parts:

1. **Set 1**: *Live* + *Generated* (N=2814): This partition has, in addition to generated data, some annotated live data in training. The motivation for this is to see if there are any improvements by folding in live data.
2. **Set 2**: *Generated* (N=3016): The domains in this partition have only generated utterances in their training data.
3. **Set 3**: *Zero-Shot Partition* (N=2392): This partition contains data for 8 domains that were not included in the training of the zero-shot model. Generative models were allowed to estimate parameters on their training data, and the embeddings k-NN model was allowed to run on it, but not retrain its embedding model.

We are particularly interested in **Set 3**, the zero-shot case, for which there is no supervised training.

4.3. Results

We use miss rate @5 to evaluate the models, i.e., the percentage of test utterances x_i for which a predicted 5-best list does not contain the correct domain y_i. While F_1 captures the performance of a model with 1-best prediction, we use miss rate @k to see how the model performs if we take top-k predictions. This is a more suitable metric compared to F_1 since our goal is to build a domain predictor which provides a top-k list of predictions. Note that for Set 3, the simple baseline models can be estimated relatively cheaply – so we show supervised numbers for those models, however in a traditional zero-shot setting this would not be allowed, so we also choose to include a 'random-guess' baseline for contrast.

Model	Set 1	Set 2	Set 3	Overall
Unigram + Bigram	**-19.8%**	-0.3%	+0.5%	-2.9%
Trigram LM	-5.6%	+8.7%	+30.7%	+12.4%
Embeddings k-NN	+467.5%	+140.6%	+218.2%	+206.4%
Random-Guess	+4182.7%	+969.3%	+1848.5%	+1646.8%
Zero-shot DNN	+97.5%	**-23.3%**	**-29.8%**	**-16.4%**

Table 2: *Miss rate @5 metric for different partitions of the test set. (Numbers are relative to unigram baseline.)*

We notice that this model tends to rank classes present during training higher than the zero-shot classes, so that they appear lower in the n-best list. This can be ameliorated by incorporating stronger contextual conditioning, for instance, a 'preferred domain' contextual feature may be powerful enough to allow domains from the zero-shot partition to displace the ones in the training partition. The model also tends to oscillate between two intent labels which both have the same utterance as a positive sample – we believe this can be further accounted for by preventing multiple correct labels from competing with one another during training.

As a side task, we also compared the zero-shot model to an n-gram based maximum entropy model baseline for intent classification within a domain, which resulted in a relative improvement in test accuracy of *8.51%* on average for the individual domains in **Set 1** and **Set 2**, although it degraded the baseline by *17.71%* relative on average for zero-shot domains. These preliminary numbers demonstrate the potential for this model class to replace thousands of domain-specific intent classifiers with disparate label sets with a single model, though new domains would currently need to be given their own models in between training cycles to maintain accuracy.

5. Conclusions and Future Work

We introduced *class attributes*, a generic framework for achieving zero-shot language understanding. We have also demonstrated a flexible neural network architecture to perform inference over classes never encountered in training. The model improves upon an averaged-embeddings baseline and performs at par with generative models while requiring much smaller vectors to be stored per domain (256-dimensional compared to 10k-100k for n-gram baselines) and also shows promise to replace all domain-specific models with a single joint model. The zero-shot architecture also provides a general framework for incorporating contextual conditioning such as personalized features into the model.

Future work can explore techniques that better map from feature spaces in one modality to another, such as Compact Bilinear Pooling popularized by [9], incorporating syntactic information into the model via subword embeddings [18], or replacing the dot product based scoring function with a learned model as has recently been popularized by adversarial methods [10]. In the context of Spoken Language Understanding, we can augment the encoders with context features and generalize them to consume ASR lattices and developer grammars [11].

6. References

[1] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *CoRR*, abs/1603.06042, 2016.

[2] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *ICML*, pages 41–48. ACM, 2009.

[3] Yun-Nung Chen, Dilek Hakkani-T, Xiaodong He, et al. Zero-shot learning of intent embeddings for expansion by convolutional deep structured semantic models. In *ICASSP*, pages 6045–6049. IEEE, 2016.

[4] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.*, 12:2493–2537, November 2011.

[5] Yann N Dauphin, Gokhan Tur, Dilek Hakkani-Tur, and Larry Heck. Zero-shot learning for semantic utterance classification. *arXiv preprint arXiv:1401.0509*, 2013.

[6] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1995.

[7] Ali Farhadi, Ian Endres, Derek Hoiem, and David Forsyth. Describing objects by their attributes. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 1778–1785. IEEE, 2009.

[8] Vittorio Ferrari and Andrew Zisserman. Learning visual attributes. In *NIPS*, pages 433–440, 2007.

[9] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. *Conference on Empirical Methods in Natural Language Processing*, 2016.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[11] Faisal Ladhak, Ankur Gandhe, Markus Dreyer, Lambert Mathias, Ariya Rastrow, and Björn Hoffmeister. Latticernn: Recurrent neural networks over lattices. 2016.

[12] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 951–958. IEEE, 2009.

[13] Tie-Yan Liu et al. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[15] Jean-Philippe Robichaud, Paul A Crook, Puyang Xu, Omar Zia Khan, and Ruhi Sarikaya. Hypotheses ranking for robust domain classification and tracking in dialogue systems. In *Interspeech*, 2014.

[16] Bernardino Romera-Paredes and Philip H. S. Torr. An embarrassingly simple approach to zero-shot learning. In *ICML*, 2015.

[17] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1842–1850, 2016.

[18] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM, 2014.

[19] Noah A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan and Claypool, May 2011.

[20] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[21] John Wieting, Mohit Bansal, Kevin Gimpel, and Karen Livescu. Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198, 2015.