Information Extraction with focus on Multi-hop Question Answering, Intent Detection and Slot Filling : A Survey

Apurva Kulkarni, Pushpak Bhattacharyya
Indian Institute of Technology Bombay, India
apurva1q2w3e@gmail.com, pb@cse.iitb.ac.in

Abstract

Information extraction (IE) is a fundamental and important problem in natural language processing. It covers a vast spectrum of tasks that have been the focus of research since the early days of natural language processing. This survey paper gives an overview of two information extraction problems - multi-hop question answering and intent detection and slot filling for dialogue state tracking. We explore the use of knowledge graph assisted deep learning models for answering multi-hop questions, which involve answering questions that require multiple steps of reasoning. For intent detection and slot filling, we explore two different types of problem statements. We first look at models for the case where intent and slot filling classes are predetermined, and the user queries are complete and independent of any ongoing dialogue. Subsequently, we discuss a versatile intent detection and slot filling system for dialogue state tracking, which can adapt to different class schemas. This system allows for handling varying types of intents and slot filling classes in dialogues from different domains. Lastly, we examine the challenge of continual learning in the context of dialogue state tracking. This involves training the model sequentially on multiple tasks while mitigating the issue of catastrophic forgetting, where the model's performance on previous tasks significantly deteriorates when learning new tasks.

1 Problem Statement

The problem statements of the three tasks are stated below.

- **Multi-hop question answering:** Given a question and context paragraphs, the task is to predict the answer span from the context, the questions requiring multiple supporting facts from different parts of the context.
- **Intent detection and slot filling:** Given an input query and the intent and slot filling classes relevant to the query, the task is to predict the correct intent class for the query and fill all possible slots using the information from the query. The following are the sub-problems associated with this problem statement.
  - **Few-shot and Zero-shot intent detection and slot filling:** Given limited training examples (few-shot) or no training examples (zero-shot) and an unseen schema for intent classes and slots, predict the most appropriate intent class and identify the correct slot values.
  - **Continual Learning:** Given training data for multiple tasks in a sequential manner, learn a model that predicts intent classes and slot values with high accuracy for each incoming task without large degradation of performance on previously learned tasks.

2 Motivation

Information extraction systems play a crucial role in today's world. Query answering systems are widely used and allow users to access any information they require instantly. This work focuses on multi-hop question answering, which aims to simulate more complex real-world questions. Virtual assistants today play an essential role in helping users with various tasks like finding flights, booking restaurants, banking, and more, and dialogue state tracking is a crucial part of such dialogue systems. With the increase in the popularity of conversational interfaces that support a large number of services in multiple domains, there is a need for dialogue systems that can effectively support these services and easily incorporate new services. Intent and slot filling are key components of a dialogue state tracking system. The ability to support new services and learn from a continuous stream of data is an essential requirement for multi-domain

dialogue systems. This is known as continual learning (CL), and it has become a crucial problem as current deep learning systems are not able to effectively retain previously learned knowledge and adapt to new information simultaneously. The main challenges in CL are catastrophic forgetting and concept drift. As a multi-domain system is trained with a continuous stream of data where new domains are introduced one at a time, the systems tend to forget previously learned knowledge and poorly perform on earlier tasks. CL has been explored in fields like computer vision and has recently been gaining attention in NLP.

3 Background

3.1 Multi-Hop Question Answering

Multi-hop questions involve reasoning over multiple supporting facts from different parts of the context to determine the answer to the question. The supporting facts generally share common entities, which link the supporting facts together and allow for reasoning to answer multi-hop questions. Hence, encoding the information in the context in the form of a knowledge graph is very useful for multi-hop question answering as it directly links the entities from different parts of the context, which are necessary for answering the question. A lot of recent research has been focussed on combining the information stored in a knowledge graph with the robustness of deep learning techniques for complex reasoning tasks like multi-hop question answering. The question answering system that we experiment with in this work is a knowledge graph assisted deep learning system, which generates embeddings of the nodes of the knowledge graph for the deep neural network answer prediction module.

3.1.1 Knowledge Graph Embeddings

Knowledge graph embedding is a technique used to allow deep learning systems to use information from knowledge graphs. Knowledge graphs are often sparse and incomplete and are difficult to integrate into a deep learning system. Knowledge graph embedding encodes the information in knowledge graphs to dense, low-dimensional vector spaces, which is compatible with deep learning architectures. These embeddings also have more global information from the knowledge graph and can make the system robust to missing links in the knowledge graph.

There are many techniques to create knowledge graph embeddings, each trying to capture specific information from knowledge graphs. One approach used in these techniques is to train a model to optimize the embedding vectors assigned to the nodes such that every triplet from the knowledge graph satisfies a specific score function in the vector space. The many translation based embedding techniques fall under this category. Another approach is to use some general embedding for the entities and infuse information from the knowledge graph to generate the knowledge graph embedding vectors. Some popular knowledge graph embedding techniques are discussed below.

- TransE - Bordes et al. (2013) first proposed a knowledge graph embedding technique, TransE, that optimizes the embedding vectors to satisfy a simple translation based score function. It was one of the first efforts that used a translation based score function which served as the inspiration for many other translation based techniques that try to capture more information in the embeddings by adding complexity to the score function used.

  In TransE, each node and relation in the knowledge graph is represented by a unique embedding vector. These vectors are randomly initialized and are updated to satisfy the translation based score function on every triplet in the knowledge graph through an optimization process. A triplet from a knowledge graph consists of the head and tail nodes that represent some entities and a relation that connects the two. The score function in TransE forces the vector representing a head node translated by the relation vector to be close to the tail node vector for all the triplets. The score and loss functions are shown in figure 1.

$$f_r(h,t) = \|\mathbf{h} + \mathbf{r} - \mathbf{t}\|_{\ell_1/\ell_2}$$

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r',t')\in S'} \max(0, \gamma + f_r(h,t) - f_{r'}(h',t'))$$

Figure 1: Score function and Loss function for TransE

The training process for TransE is as follows. First, all triplets from the knowledge graph are extracted and stored as the golden triplets set. Then, a set of corrupted or negative triplets is created by taking the original triplets and replacing one node in each triplet with a random node in the knowledge graph. This process of creating corrupted triplets could result in false negatives, so some techniques to reduce this have been proposed in later works. Embedding vectors for nodes and relations in the knowledge graph are randomly initialized. The loss function chosen (figure 1) is optimized using stochastic gradient descent iterating over all golden and corrupted triplets. This process generates vector embeddings that satisfy the score function for golden triplets and not for incorrect triplets because of how the loss function in figure 1 is constructed.

TransE is the most popular translation based embedding technique because of its simplicity and effectiveness, but it has a few drawbacks. One of the main drawbacks of the simple score function is described in the following situation. If two triplets share the same relation and tail node with different head nodes, then the score function will try to update the vectors of the two head nodes to be close to each other. However, if the same head nodes are each connected to different tail nodes by some other relation in the knowledge graph, then it would require the head node vectors to be different according to the score function. For example, if 'Modi' and 'India' are related by 'Prime Minister' relation, 'Nehru' and 'India' are related by 'Prime Minister' relation, then this would require embeddings of 'Modi' and 'Nehru' to be close according to TransE scoring function, but another relation like 'Political Party' would require them to be far.

- TransH - Feng (2014) proposed TransH, a translation based embedding technique similar to TransE but with one added complexity to the score function. It aimed to rectify the above mentioned drawback of TransE caused because of the simplicity of the score function. TransH adds a step in the score function to distinguish between the different relations in the knowledge graph. This is achieved by projecting all embedding vectors to a hyperplane specific to each relation before the translation operation in the score function. Each relation is assigned two vectors, one is its embedding vector like in TransE, and another to define its unique hyperplane. These are represented as d_r and w_r respectively in figure 3. h, r and t represent the head node, relation and tail node embeddings respectively. f_r is the score function and L is the loss function.

  The training process is the same as that in TransE. TransH is able to improve on the drawback of TransE since if two head nodes are connected to the same tail node by the same relation; then their embedding vectors need not be the same; only the projection of the head vectors onto the hyperplane specific to that relation need to be the same. This allows TransH embeddings to capture more information from the knowledge graph.

- Many other translation based embedding techniques including TransR (Lin et al., 2015) and TransD (Ji et al., 2015) followed the above. In Lin et al. (2015), the authors observed that entities could have multiple aspects and cannot capture all relations in the same space. It modeled entities and relations in distinct spaces, and the scoring function projected entities to the relation space before the translation operation. Other techniques like TransD (Ji et al., 2015), ComplEx (Trouillon et al., 2016), DistMult (Yang et al., 2014) all generate embeddings on the same principle as above and employ more complex score functions. Later works introduced neural network layers in the score function computation, one of which is discussed below.

- ConvE - The ConvE embedding technique Dettmers et al. (2018) uses a score function that performs 2D convolutions over the knowledge graph embeddings. The training process is similar to all the techniques discussed before. The following figure 2 shows the operations in the score function

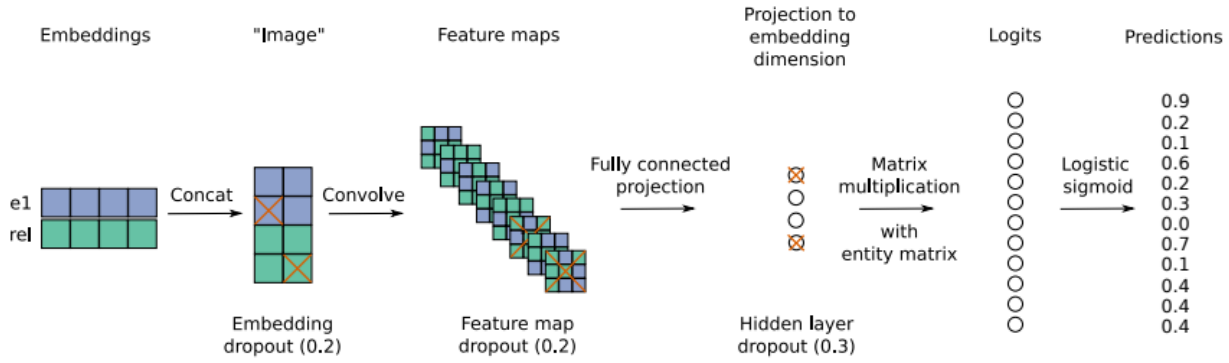

Figure 2: ConvE embedding score function
(Dettmers et al., 2018)

$$\mathbf{h}_\perp = \mathbf{h} - \mathbf{w}_r^\top \mathbf{h} \mathbf{w}_r,\ \mathbf{t}_\perp = \mathbf{t} - \mathbf{w}_r^\top \mathbf{t} \mathbf{w}_r$$

$$f_r(h,t) = \|\mathbf{h}_\perp + \mathbf{d}_r - \mathbf{t}_\perp\|_2^2$$

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r',t')\in S'} \max(0, \gamma + f_r(h,t) - f_{r'}(h',t'))$$

Figure 3: Score function and Loss function for TransH

computations of ConvE.

- The other class of embedding techniques takes general embeddings representing entities in the knowledge graph and infuses information of the knowledge graph connections into them. Graph Convolutional Networks (Kipf and Welling, 2016) and Graph Neural Networks are examples of this, which require general input embeddings and output knowledge graph aware embeddings. The former technique, GCN, is used in our work and is described in detail in the next section.

3.1.2 Graph Convolutional Networks

Graph Convolutional Network (GCN) Embedding is an embedding technique that uses deep neural networks with graph convolution layers to generate embeddings from a knowledge graph. Unlike the translation based embedding techniques discussed in the previous section, GCN embeddings require some general initial embedding that represents each node, like Glove or BERT embeddings. These input embeddings and all knowledge graph triplets are passed as input to the GCN model, which generates embeddings infused with information from the knowledge graph.

The core operation in the GCN is the graph convolution. It takes inspiration from the convolution operation, which is a weighted average of neighboring matrix values. Similarly, the graph convolution operation averages the corresponding values of connected neighbor nodes for the output corresponding to each node. The process is described in detail below.

The input embeddings for every node and the knowledge graph triplets are stored. From the knowledge graph triplet information, the adjacency matrix is constructed. If 'N' is the number of nodes in the graph, then the adjacency matrix is an 'NxN' matrix in which each node corresponds to one row and one column. An entry in the adjacency matrix is one if the nodes corresponding to its row and column are connected in the knowledge graph; otherwise, it is zero. The graph convolution operation is shown in figure 4. The first step in it is the product of a weight matrix, the matrix of input embedding vectors, and the adjacency matrix. The result of this is a matrix of vectors formed by the weighted average of vectors corresponding to connected neighbor nodes. In the figure, A is the adjacency matrix, H^i is the input embedding matrix, W is the weight matrix and H^{i+1} is the output embedding matrix. The result is passed through the activation function to get the output of the graph convolution layer. Multiple graph convolution layers stacked together form a GCN.

$$H^{[i+1]} = \sigma(W^{[i]} H^{[i]} A^*)$$

Figure 4: Graph Convolution Operation

As described previously, the output of a graph convolution layer for a particular node is a weighted average of the input vectors corresponding to nodes connected to that node in the knowledge graph. This way, the output embeddings of the GCN model are infused with connection information from the knowledge graph. Subsequent deep learning models can then use the original input embeddings and the final GCN embeddings, allowing them to benefit from the knowledge graph information.

3.1.3 HotpotQA Dataset

HotpotQA is a question answering dataset with open-domain multi-hop questions. The dataset was collected by the authors of (Yang et al., 2018). The purpose of this dataset was to train question answering systems to answer more challenging questions and perform complex reasoning since most datasets like SQUAD have questions with direct answers and do not address many challenges of complex real-world questions. The dataset contains 113k Wikipedia-based question-answer pairs and context paragraphs for each pair. The questions require finding and reasoning over multiple supporting paragraphs to answer and are not constrained to any pre-existing knowledge bases.

The dataset contains two main types of questions in terms of reasoning, 'bridge' type and 'comparison' type. 'Bridge' type questions require answering multiple questions in order, where the answer to one part of the question is needed to answer the following parts. For example, the question "Who is the Prime minister of the country with the second largest population?" requires answering 'India' for the country, which is necessary to answer the first part. An example of 'comparison' type is "Which book is older, Harry Potter or Oliver Twist?" where two questions about when the books were written are parallelly answered, and then the final answer is determined. Table 1 shows an example from the HotpotQA dataset.

3.2 Intent Detection and Slot Filling

3.2.1 ATIS Dataset

The ATIS dataset (Airline Travel Information Systems) is a dataset consisting of audio recordings and corresponding manual transcripts about humans asking for flight information on automated airline travel inquiry systems. It is annotated for intent and slot filling tasks, with 17 unique intent classes. Examples of instances from the dataset is shown in tables 2 and 3.

3.2.2 SNIPS Dataset

SNIPS (Coucke et al., 2018) is an open domain dataset for intent and slot filling tasks with 16000 crowdsourced queries distributed among 7 user intents. The intent classes are SearchCreativeWork (e.g. Find me the I, Robot television show), GetWeather (e.g. Is it windy in Boston, MA right now?), BookRestaurant (e.g. I want to book a highly rated restaurant in Paris tomorrow night), PlayMusic (e.g. Play the last track from Beyoncé off Spotify), AddToPlaylist (e.g. Add Diamonds to my roadtrip playlist), RateBook (e.g. Give 6 stars to Of Mice and Men), and SearchScreeningEvent (e.g. Check the showtimes for Wonder Woman in Paris), covering multiple different domains. Further annotated examples are shown in tables 4 and 5.

3.2.3 Schema Guided Dataset

The Schema Guided Dataset (SGD) (Rastogi et al., 2020) is a dataset consisting of 20k annotated multi-domain, task-oriented conversations between a human and a virtual assistant spanning 26 services belonging to 16 domains. The dataset has been annotated for intent and slot filling tasks. This dataset aims to simulate the real world requirement of adapting systems to different domains. To simulate this, it introduces the complexity of varying intent and slot classes in instances from different domains, with zero shot test set evaluation.

The core feature of this dataset is the provision of schemas for every service in the dataset. A schema for a service is the combination of possible intents and slots that can be associated with an utterance from a dialogue of that service, along with descriptions of the services and classes. This fixes the slot combinations for a service and ensures that the slot combinations are realistic and supported by actual service APIs. Each schema has the name and description of the service it corresponds to. It also contains the list of possible intent and slot classes for the service. Each class has a name and a de-

Question	The Oberoi family is part of a hotel company that has a head office in what city?
Relevant Context Paragraphs	***Paragraph 1**: **The Oberoi family** is an Indian family that is famous for its involvement in hotels, namely through **The Oberoi Group.*** ***Paragraph 2**:**The Oberoi Group** is a hotel company with its head office in **Delhi**. Founded in 1934, the company owns and operates 30+ luxury hotels and two river cruise ships in six countries.*
Answer	Delhi

Table 1: Example from HotpotQA

Query	Intent Class
show me the fares from dallas to san francisco	atis-airfare
please give me flights available from baltimore to philadelphia	atis-flight
what ground transportation is there in atlanta	atis-ground service

Table 2: Examples of ATIS intent classes

Query	Slot filling tags
show me the fares from dallas to san francisco	O-O-O-O-O-Bfromloc.cityname-O-B-toloc.cityname-Itoloc.cityname
please give me flights available from baltimore to philadelphia	O-O-O-O-O-O-Bfromloc.cityname-O-Btoloc.cityname
what ground transportation is there in atlanta	O-O-O-O-O-O-Bcityname

Table 3: Examples of ATIS slot labelling

Query	Intent Class
i want to listen to seventies music	PlayMusic
show me the picture creatures of light and darkness	SearchCreativeWork
i d like to go to the popular bistro in oh	BookRestaurant

Table 4: Examples of SNIPS intent classes

Query	Slot filling tags
i want to listen to seventies music	O-O-O-O-O-Byear-O
show me the picture creatures of light and darkness	O-O-O-Bobjecttype-Bobjectname-Iobjectname-Iobjectname-Iobjectname-Iobjectname
i'd like to go to the popular bistro in oh	O-O-O-O-O-O-O-Bsort-Brestauranttype-O-Bstate

Table 5: Examples of SNIPS slot labelling

scription that defines that class. The schema also has information about the various constraints, like the required slots for calling the class or the possible values of the class. An example of a schema is given below.

The specific intent and slot filling classes of a particular training example together is referred to as a schema. For each intent and slot filling class in the schema, a text description is provided that defines that class. The dataset also evaluates slot filling accuracy on all utterances in a dialogue together, which is called joint goal accuracy.

3.3 Continual Learning

The Continual learning problem aims to learn tasks sequentially from a continuous stream of data without the catastrophic forgetting of previously learned knowledge. Specifically, the goal is to sequentially learn a model from a set of tasks T which are trained on sequentially. The model has to maximize performance on the current task it is trained on while minimizing catastrophic forgetting on all

Service	**Service Name:** Payment **description:** Digital wallet to make and request payments
Slots	**Name:** Account type **description:** Source of money to make payment **Name:** Amount **description:** Amount of money to transfer on request **Name:** Contact Name **description:** Name of contact for transaction
Intents	**Name:** Make Payment **description:** Send money to your contact **Name:** Request Payment **description:** Request money from a contact

Table 6: Example of SGD schema

the previous tasks it has been trained on.

There are many properties that have to be observed when building by CL models. The first is the minimization of catastrophic forgetting, as discussed above. Then there is forward transfer, where the model uses previous knowledge to improve performance on future tasks, and backward transfer, where the model's performance improves on older tasks after training on new tasks. Next is memory capacity, which is relevant for methods like rehearsal-based training, where some examples from older tasks are stored and used while training new tasks. Finally, there is the plasticity of the model, which refers to the ability to quickly learn from new tasks after training on a large number of tasks. The following sections detail the different approaches to continual learning.

3.3.1 Rehearsal Methods

Rehearsal techniques involve retaining some training examples from prior tasks and reusing them during future training. The number of examples that can be retained is problem specific and is fixed to compare all rehearsal methods. Some approaches train models to generate these examples, which are known as pseudo-rehearsal methods. Pseudo-rehearsal methods use models like generative adversarial networks or generative autoencoders. Another rehearsal method is to constrain the gradient updates so that the loss of the samples in memory never increases. Details of a few methods are discussed below.

One basic popular method used to collect replay during training is reservoir sampling. In this method, all incoming samples are retained until the memory buffer or retained sample set reaches the maximum allowed size. After this, each incoming training sample is exchanged with a sample in the memory reservoir with a probability proportional to the number of training samples seen before the current sample. The details for this method are given in 1 below, as described in Riemer et al. (2019). Here M is the maximum memory buffer capacity, N is the total number of training examples seen before the current example, and (x,y) is the training input sample and label.

Procedure RESERVOIR(M, N, x, y)**:**
  if $M > N$ **then**
    $M[N] \leftarrow (x, y)$
  end
  else
    $j = randomInteger(min = 0, max = N)$
    if $j < M$ **then**
      $M[j] \leftarrow (x, y)$
    end
  end
return M

Algorithm 1: Reservior Sampling

Gradient Episodic Memory (GEM) (Lopez-Paz and Ranzato, 2022) is another very popular method for continual learning. The main idea of this technique is to modify the gradient update to ensure that the loss of the model trained on the examples from the memory buffer is less than the loss of the model trained up to the previous task. This allows for positive back transfer. The disadvantage of this method is that it uses a quadratic programming solver that scales with the number of parameters of the model and is impractical for large language models.

3.3.2 Regularization Methods

Regularization methods involve slowing down the learning of parameters important for previous tasks by regularizing the loss. These methods rely on a fixed model capacity with an additional loss term that aids knowledge consolidation while learning subsequent tasks or data distributions. Techniques that penalize changes to parameters that are deemed important for previous tasks during CL fall under this category. Elastic Weight Consolidation (EWC) (Kirkpatrick et al., 2016) is an example of this kind of method, where forgetting is reduced by regularizing the loss to prevent large updates to parameters that are important to previous tasks. The main disadvantage of these methods is that they reduce the plasticity of the models to reduce forgetting, which hampers learning on new tasks as the number of tasks in the CL process increases. Hence, these methods are not feasible for cases where the total number of tasks is very large.

3.3.3 Architectural methods

Architectural methods involve introducing changes to model architecture dynamically to prevent forgetting. Progressive Networks and Dynamically Expandable Networks fall under this category. Many of these methods fully eliminate catastrophic forgetting as they freeze parameters learnt on previous tasks and introduce new parameters for each subsequent task. The main drawback of these methods is the increase in the model size which makes them infeasible for CL with longer sequences of tasks.

- **Progressive Networks:** Progressive Neural Networks (PNN) (Rusu et al., 2022) use a dynamic architecture where catastrophic forgetting is prevented by instantiating a new neural network column for each task being solved. They also have lateral connections between columns to transfer features learned from previous tasks. During training, only the newly introduced column is kept trainable, and the weights of other columns are frozen. The main advantage of this method is that it ensures zero catastrophic forgetting and the lateral connections allow for positive forward transfer to newer tasks. The main drawback of the PNN architecture is the continuous increase in the number of model parameters with each new task which makes it infeasible to use PNNs when the number of tasks is very large.
- **Adaptor CL:** Adaptor CL is an architectural method used by Madotto et al. (2020) for CL for DST. The model uses residual adaptors with a transformer model for CL. Residual adapters are trainable parameters added on top of each transformer layer, which steer the output distribution of a pre-trained model without modifying its original weights. For continual learning, this model learns different adapter weights for each task without modifying the original weights of the transformer model. This eliminates catastrophic forgetting at the cost of increasing model parameters with each new task.

3.3.4 Meta-learning based methods

Meta-learning or learning-to-learn involves learning generic knowledge, given a small set of training examples and numerous tasks, and quickly adapting to new tasks. In the following sections, we discuss Model-Agnostic Meta-learning (MAML) and variations of this method for continual learning.

3.3.5 Model-Agnostic Meta-learning (MAML)

Model-Agnostic Meta-learning (MAML) (Finn et al., 2017a) is a meta-learning method designed to be compatible with any model architecture that uses gradient descent for any kind of learning task. The method was created with the goal of fast, few-shot adaptation on new tasks or for new domains. In this method, the training happens in two phases, the first is the meta-training phase in which corresponding meta parameters are learnt, and the second is the fine-tuning phase on the few-shot training examples.

The meta parameters include the model's initial weights and learning rates, which are updated in the meta-training phase. The objective of the meta training phase is to learn these meta parameters by training on data from multiple tasks, allowing the meta parameters to capture the common features between all tasks of the particular type of problem. Learning meta-parameters broadly applicable to the tasks of a particular type of problem allows for faster fine-tuning on new tasks with minimal data. The algorithm for MAML (Finn et al., 2017a) is discussed below.

In algorithm 2, $p(T)$ is the distribution over tasks, and α, β are learning rates for inner and meta updates. θ is the model initial weights tensor and L is the loss function. The algorithm trains on

Procedure MAML$(p(T), \alpha, \beta, N)$**:**
  randomly initialize θ
  for *all* T_i **do**
    Sample batch of tasks $T_i \sim p(T)$
    for $n \leftarrow 1$ **to** N 1 **do**
      Evaluate $\nabla_\theta L_{T_i}(f_\theta)$
      $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta)$
    end
    Update $\theta = \theta - \beta \nabla_\theta L_{T_i}(f_{\theta_i'})$
  end
return θ

Algorithm 2: Model-Agnostic Meta-learning

a batch of selected tasks in the inner loop. θ_i' is the parameter tensor after the inner loop training. Then the original meta parameters θ, which are the meta model initial parameters, are updated in the outer loop with the gradient of the loss on the batch evaluated with θ_i' as the model parameters. This essentially trains the model initial parameters to allow for faster convergence during fine tuning on any task. The learnt initial parameters also give good performance for related tasks not seen during meta training.

3.3.6 Online Meta-learning (OML)

Online Meta-learning (OML) (Finn et al., 2017b) is a variant of the MAML algorithm for the continual learning setting. Instead of learning initial weights for the entire model for a set of tasks like in MAML, it splits the model into representation layers and adaptation layers and the meta learning objective tries to learn weights for the representation layers that reduce catastrophic forgetting. We use a modification of the algorithm in Finn et al. (2017b), the details of the algorithm are discussed below in algorithm 3.

In algorithm 3, $p(T)$ is the distribution over tasks, and α, β are learning rates for inner and meta updates. θ is the representation model initial weights tensor, W is the adaptation model initial weights tensor, and L is the loss function. θ_i' is the parameter tensor after the inner loop training. Then the original meta parameters θ, which are the meta model initial parameters, are updated in the outer loop with the gradient of the loss on the batch evaluated with θ_i' as the model parameters. This process simulates the fine tuning of the adaptation layers in the inner loop, and the meta update changes the representation layer weights in a way that improves fine tuning in the next iteration. The learnt initial

Procedure OML$(p(T), \alpha, \beta, N)$**:**
  randomly initialize θ
  for *all* T_i **do**
    randomly initialize W
    Sample batch of tasks $T_i \sim p(T)$
    Sample a sequence of k examples S_k from T_i
    $W_0 = W$
    for $j \leftarrow 1$ **to** k 1 **do**
      $X_j = S_k[j]$ Evaluate $\nabla_\theta L_{T_i}(f_\theta(X_j))$
      $\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta(X_j))$
    end
    Sample S_{test} from T_i Update $\theta = \theta - \beta \nabla_\theta L_{T_i}(f_{\theta_i'}(S_{test}))$
  end
return θ

Algorithm 3: Online Meta-learning

parameters also give good performance for related tasks not seen during meta training.

3.3.7 Look-ahead Model-Agnostic Meta-learning (La-MAML)

Look-ahead Model-Agnostic Meta-learning (La-MAML) (Gupta et al., 2020) proposes a modification on OML to eliminate the separate meta learning stage and introduces training procedure with both meta training and fine tuning combined together. It also introduces trainable learning rates for model parameters as a part of the meta training step. Each parameter of the model is assigned a separate learning rate which is learnt by meta training as described in algorithm 4. Also a memory buffer is used to store training samples from previous tasks for meta training.

In algorithm 4, $p(T)$ is the distribution over tasks, and α, β are learning rates for inner and meta updates. α is a tensor of learning rates α^j for each model parameter. θ is the model initial weights tensor and L is the loss function. Unlike the MAML algorithm, the learning rates which also get updated in the meta update step.

4 Multi-Hop Question Answering

As discussed in the previous chapter, the requirement of complex reasoning in multi-hop question answering makes knowledge graph assisted deep learning models a suitable choice for this task. There are many recent works that use knowledge graph embeddings and deep neural networks for

Procedure LaMAML($p(T), \alpha, \beta_1, \beta_2, N$):
  randomly initialize θ
  for *all* T_i **do**
    Sample current task $T_i \sim p(T)$
    for $n \leftarrow 1$ **to** N 1 **do**
      Sample from memory buffer
      Evaluate $\nabla_\theta L_{T_i}(f_\theta)$
      $\theta_i' = \theta - \alpha^j \nabla_\theta L_{T_i}(f_\theta)$
    end
    Update $\alpha^j + 1 = \alpha^j - \beta_1 \nabla_\theta L_{T_i}(f_{\theta_i'})$
    Update $\theta^j + 1 = \theta - \beta_2 \nabla_\theta L_{T_i}(f_{\theta_i'})$
  end
return θ

Algorithm 4: Look-ahead MAML

multi-hop question answering. Saxena et al. (2020) and Huang et al. (2019) are two such works with very similar architectures. While they try to employ knowledge graphs and deep learning for multi-hop question answering, the problem statement in these works is slightly different. Instead of context paragraphs, the question answering system answers questions by referring to a knowledge graph. The system works under the assumption that the answer is a node in the referred knowledge graph. We describe the system in Saxena et al. (2020) in detail below.

Figure 5 shows the system architecture proposed in Saxena et al. (2020). The system generates knowledge graph embeddings for all nodes using the ComplEx embedding technique. The questions are then mapped to the space of head node embeddings and relation embeddings, respectively, using neural networks. Given an input question, the system maps it to the two embedding spaces. The system assumes that there is a triplet in the knowledge graph corresponding to the question and that the head node and relation information will be in the question, with the tail node being the answer. The system then ranks candidate answer nodes using the ComplEx embedding scoring function and selects the highest scoring tail node a the output answer.

The model proposed in Zheng and Kordjamshidi (2020) is more relevant to our problem statement, and our work is based on the above model architecture. Their proposed system creates a knowledge graph from the context passages and then uses GCN embeddings and deep neural networks to predict the answer. The knowledge graph is created by extracting phrases and relations from the context using semantic role labeling. The nodes are represented by Glove embeddings and are passed to the GCN module along with the input question. The GCN embeddings are concatenated with the original input embeddings and are passed to the answer span prediction module. By creating a knowledge graph from the context and embedding it, the supporting facts that can be distant in the context are connected nodes in the knowledge graph.

5 Intent Detection and Slot Filling

In this section, we cover the different approaches that exist for intent detection and slot filling, for both the fixed class and variable class cases.

5.1 Intent Detection and Slot Filling for fixed classes

Query understanding has been an important topic of research for the last several years. Intent detection plays an important role in providing a satisfying user experience. In Sreelakshmi et al. (2018) the authors propose a deep learning-based framework using Bi-Directional Long Short-Term Memory (BLSTM) for intent detection. The model takes in word embeddings of the input and learns features to identify the intent of the query. They also discuss use of an enrichment technique of embeddings to ensure semantic correctness of the embeddings. Kumar et al. (2019) adapts BERT transformer model and fine tune the BERT model with domain specific training for e-commerce domain queries and demonstrate the performance of pretraining BERT model on a large corpus and then performing domain specific fine tuning.

For the ATIS and SNIPS dataset, Chen et al. (2019) proposed a BERT based architecture that achieved state of the art results. The key feature of this architecture is the shared model and combined training of intent detection and slot filling. The ATIS dataset is a very small dataset, and the proposed system outperforms conventional BERT based architectures. The authors also explore the use of CRF for modeling slot filling. For the joint training of intent detection and slot filling, different learning rates are used for each task, which are tunable parameters. Wang et al. (2018) proposed a BiLSTM based model for intent and slot detection and is one of the best performing models on the ATIS and SNIPS dataset.

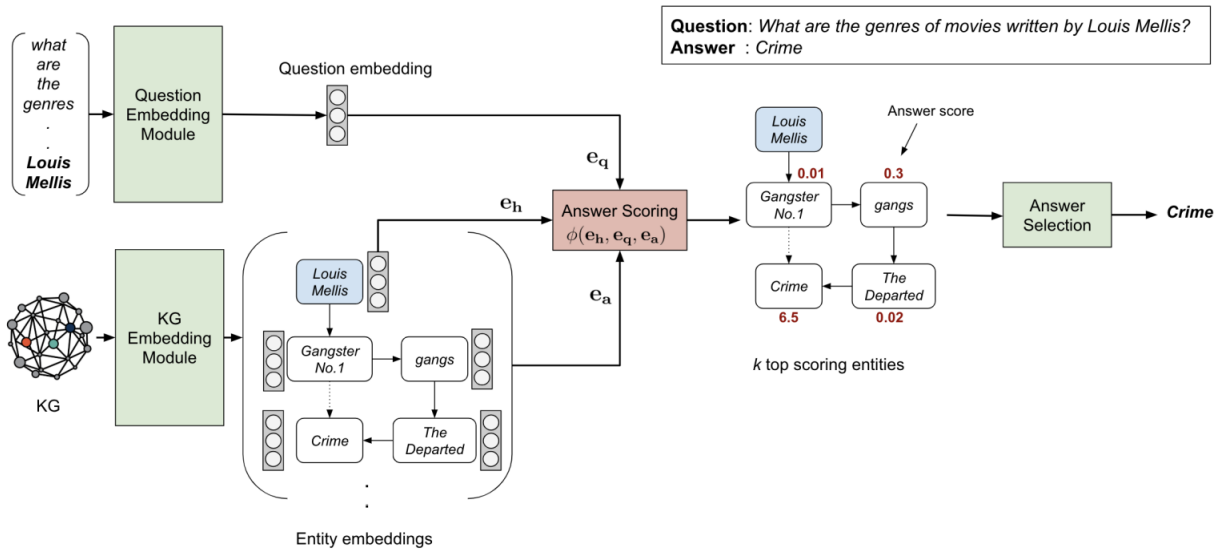

Figure 5: Knowledge graph assisted deep learning system architecture (Saxena et al., 2020)

5.2 Intent Detection and Slot Filling for variable classes

For the case of variable intent and slot classes, the Schema Guided Dataset (SGD) is the only popular dataset, covering a large number of domains and class schemas. Rastogi et al. (2020) and Feng et al. (2021a) propose adaptable model architectures to account for varying classes. Both have a similar approach to incorporating the above. The schema provided has a description for each intent and slot class. The system passes these descriptions through BERT and generates schema embeddings for each class. These embeddings are combined with traditional prediction neural networks to allow for intent detection and slot filling variable classes.

Zhang et al. (2021) and Noroozi et al. (2020) propose robust and efficient architectures and reduce repetitive computations. Noroozi et al. (2020) proposes to split the utterance and schema encodings and concatenate these embeddings in the decoder. Since the schema is known beforehand, all the schema embeddings are computed and stored in memory before the training process. This reduces the computation and speeds up the model. Feng et al. (2021b) uses a sequence to sequence problem architecture for dialogue state tracking. It uses an utterance-schema attender to combine the outputs of the utterance and schema encoders and uses an LSTM decoder to generate the decoder states.

5.3 Continual learning in dialogue state tracking

There are many different approaches to continual learning, each with different advantages and disadvantages. The first kind is called rehearsal, where training examples from previous tasks are reintroduced in the training of later tasks. A memory buffer of a fixed maximum size is allowed for these methods in which they store and use training samples from previously trained tasks in different ways. Reservoir sampling is a simple method used in many CL methods that use a memory buffer. Lopez-Paz and Ranzato (2022) and Chaudhry et al. (2018) are popular methods that use rehearsal along with a constrained optimization of the loss function and try to minimize catastrophic forgetting. Kemker and Kanan (2017) and Shin et al. (2017) uses generative models to create examples from previous tasks for rehearsal.

Regularization based approaches are popular for continual learning. Kirkpatrick et al. (2016) is an example of this kind of method, where forgetting is reduced by regularizing the loss to prevent large updates to parameters that are important to previous tasks. Aljundi et al. (2018), Nguyen et al. (2017) and Zenke et al. (2017) use regularization-based methods to estimate the importance of each model parameter for previous tasks and penalize changes to each parameter.

Architectural methods involve introducing changes to model architecture dynamically to prevent forgetting. Rusu et al. (2022) proposed the pro-

gressive model architecture for continual learning, which transfers knowledge from previous tasks to improve convergence speed. Progressive networks naturally accumulate experiences and are immune to catastrophic forgetting by design. Adaptor CL is an architectural method used by Madotto et al. (2020) for CL for DST. The model uses residual adaptors with a transformer model for CL. The drawback of these approaches is the growth of the number of model parameters with the introduction of new tasks.

Meta-learning approaches are another strong candidate for continual learning with works like Finn et al. (2017b), Gupta et al. (2020), and Zou and Lin (2022). Finn et al. (2017b) proposes online-aware meta-learning (OML), which is a meta-learning technique that reduces interference and overwriting of previous knowledge during training updates. Gupta et al. (2020) proposes Look-Ahead-MAML, a modification of OML that learns different learning rates for each parameter in the model as a part of the meta-training step, and also merges the meta training and fin-tuning processes together. While meta-learning techniques lead to good generalization and less hand engineering for training, it is computationally expensive and requires careful design.

The problem of CL in dialogue systems has been explored in Madotto et al. (2020) and Liu et al. (2021). Liu et al. (2021) proposes the knowledge preservation network architecture for CL and performs experiments using the SGD Dataset for slot filling. Madotto et al. (2020) proposed a CL method using an adaptor-based approach given by Houlsby et al. (2019). The model uses residual adaptors with a transformer model for CL. Residual adapters are trainable parameters added on top of each transformer layer, which steer the output distribution of a pre-trained model without modifying its original weights. For continual learning, this model learns different adapter weights for each task without modifying the original weights of the transformer model. This eliminates catastrophic forgetting at the cost of increasing model parameters with each new task. Zhu et al. (2022) introduces a method of training a prompt for each task in the continual learning process while fixing the pre-trained model parameters. This prevents forgetting of previous knowledge and task-specific prompts can be used to evaluate the performance on each task.

6 Summary

In this paper, we explore two important information extraction problems, multi-hop question answering and dialogue state tracking for task-oriented dialogue systems. For multi-hop question answering, we first discuss knowledge graphs which are important structures that allow question answering systems to reason over multiple supporting facts. We then explore how knowledge graphs can be used along with deep learning systems with the help of knowledge graph embedding techniques. We discuss many specific KG embedding techniques and multi-hop question answering architectures that use KG embedding and deep neural networks.

For the problem of intent detection and slot filling, we first discuss the fixed class, single domain case. We discuss system architectures for intent detection and slot filling for datasets like ATIS and SNIPS where the total number of intent classes and slots are fixed. We then look at the case of multiple domains and varying intent classes and slots. We discuss the schema-guided approach to dialogue state tracking, which allows models to deal with variable classes. We discuss specific architectures for the SGD Dataset which introduces the complexity of zero shot prediction on data from unseen services. We finally discuss the problem of continual learning for dialogue state tracking and describe different approaches used to reduce catastrophic forgetting in models trained on tasks sequentially.

References

Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. 2018. Memory aware synapses: Learning what (not) to forget.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. 2013.

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. 2018. Efficient lifelong learning with a-gem.

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

Alice Coucke, Alaa Saade, Adrien Ball, Théodore Bluche, Alexandre Caulier, David Leroy, Clément Doumouro, Thibault Gisselbrecht, Francesco Caltagirone, Thibaut Lavril, Mael Primet, and Joseph Dureau. 2018. Snips voice platform: an embedded

spoken language understanding system for private-by-design voice interfaces.

T Dettmers, Pasquale Minervini, P Stenetorp, and Sebastian Riedel. 2018. Convolutional 2d knowledge graph embeddings.

Jianlin Feng. 2014. Knowledge graph embedding by translating on hyperplanes.

Yue Feng, Yang Wang, and Hang Li. 2021a. A sequence-to-sequence approach to dialogue state tracking. pages 1714–1725.

Yue Feng, Yang Wang, and Hang Li. 2021b. A sequence-to-sequence approach to dialogue state tracking.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017a. Model-agnostic meta-learning for fast adaptation of deep networks.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017b. Model-agnostic meta-learning for fast adaptation of deep networks.

Gunshi Gupta, Karmesh Yadav, and Liam Paull. 2020. La-maml: Look-ahead meta learning for continual learning.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp.

Xiao Huang, Jingyuan Zhang, Dingcheng Li, and Ping Li. 2019. Knowledge graph embedding based question answering. pages 105–113.

Guoliang Ji, Shizhu He, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Knowledge graph embedding via dynamic mapping matrix. pages 687–696.

Ronald Kemker and Christopher Kanan. 2017. Fearnet: Brain-inspired model for incremental learning.

Thomas Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2016. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114.

Mukul Kumar, Youna Hu, Will Headden, Rahul Goutam, Heran Lin, and Bing Yin. 2019. Shareable representations for search query understanding.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. *Proceedings of AAAI*, 29:2181–2187.

Qingbin Liu, Pengfei Cao, Cao Liu, Jiansong Chen, Xunliang Cai, Fan Yang, Shizhu He, Kang Liu, and Jun Zhao. 2021. Domain-lifelong learning for dialogue state tracking via knowledge preservation networks. pages 2301–2311.

David Lopez-Paz and Marc'Aurelio Ranzato. 2022. Gradient episodic memory for continual learning.

Andrea Madotto, Zhaojiang Lin, Zhenpeng Zhou, Seungwhan Moon, Paul Crook, Bing Liu, Zhou Yu, Eunjoon Cho, and Zhiguang Wang. 2020. Continual learning in task-oriented dialogue systems.

Cuong Nguyen, Yingzhen Li, Thang Bui, and Richard Turner. 2017. Variational continual learning.

Vahid Noroozi, Yang Zhang, Evelina Bakhturina, and Tomasz Kornuta. 2020. A fast and robust bert-based dialogue state tracker for schema-guided dialogue dataset.

Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. 2020. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:8689–8696.

Matthew Riemer, Ignacio Cases, Robert Ajemian, Miao Liu, Irina Rish, Yuhai Tu, and Gerald Tesauro. 2019. Learning to learn without forgetting by maximizing transfer and minimizing interference.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2022. Progressive neural networks.

Apoorv Saxena, Aditay Tripathi, and Partha Talukdar. 2020. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. pages 4498–4507.

Hanul Shin, Jung Lee, Jaehong Kim, and Jiwon Kim. 2017. Continual learning with deep generative replay.

K Sreelakshmi, P Rafeeque, S Sreetha, and E Gayathri. 2018. Deep bi-directional lstm network for query intent detection. *Procedia Computer Science*, 143:939–946.

T Trouillon, J Welbl, Sebastian Riedel, Eric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018. A bi-model based rnn semantic frame parsing model for intent detection and slot filling. pages 309–314.

Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases.

Zhilin Yang, Peng Qi, Saizheng Zhang, Y. Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher Manning. 2018. Hotpotqa: A dataset for diverse, explainable multi-hop question answering.

Friedemann Zenke, Ben Poole, and Surya Ganguli. 2017. Continual learning through synaptic intelligence.

Yang Zhang, Vahid Noroozi, Evelina Bakhturina, and Boris Ginsburg. 2021. Sgd-qa: Fast schema-guided dialogue state tracking for unseen services.

Chen Zheng and Parisa Kordjamshidi. 2020. Srlgrn: Semantic role labeling graph reasoning network. pages 8881–8891.

Qi Zhu, Bing Li, Fei Mi, Xiaoyan Zhu, and Minlie Huang. 2022. Continual prompt tuning for dialog state tracking.

Xiaohan Zou and Tong Lin. 2022. Efficient meta-learning for continual learning with taylor expansion approximation.