A Study of Natural Language Understanding

Tianyi Sun

Maria Gini
University of Minnesota, Twin Cities

Table of Contents

1 Abstract

In the current generation of Natural Language Processing (NLP), in order to make language model robust on understanding and reasoning, we need to train it on a large amount of data samples. However, sometimes it is hard to collect a huge amount of data, for example, time series data for tracking people's emotional responds to COVID-19. Thus, to improve language model's comprehensive and reasoning abilities while training on a small number of text samples, we further explore the syntax use and semantic parsing as well as taking advantage of the mathematical logic rules to form an internal logic for language model. But it is true that the structure of texts is variable according to the domain and environment, so it is need to combine the internal logic with the use of Meta-Learning to follow the factuality of texts.

2 Introduction

By definition, language is the principal method of human communication, consisting of words used in a structured and conventional way and conveyed by speech, writing, or gesture. NLP enables computers to understand and process human language.

Currently the biggest questions is that can human beings communicate with computers in human beings' natural languages? One big challenge of addressing this is that human speech is unstructured data and ambiguous in nature, however, computers need structured data.

2.1 History

Based on the main concerns at that period of time the history of NLP can be divided into four phases.

The first phase, **Machine Translation Phase**, is from late 1940s to late 1960s. In early 1950s, following Booth and Richens' investigation and Weaver's memorandum on machine translation, NLP research started. In 1954, automatic translation from Russian to English demonstrated in the Georgetown-IBM experiment. In 1961, the high lighted work on Machine Translation of Languages and Applied Language analysis presented in Teddington International Conference.

The second phase is from late 1960s to late 1970s, which is mainly about **world knowledge** and the construction and manipulation of **meaning representations**. In 1961, a BASEBALL question-answering system with a restricted input and a simple language processing was developed. In 1968, a much advanced system, which could inference on the knowledge base in interpreting and responding to language input was described.

The third phase is from late 1970s to late 1980s, researchers moved to **the use of logic for knowledge representation and reasoning**, due to the failure of building practical system in last phase. Some practical tools like parsers, e.g. Alvey Natural Language Tools, and more operational and commercial systems for database query are invented. The grammatical-logical approach and lexicon made general-purpose sentence processors more powerful, e.g. SRI's Core Language Engine and Discourse Representation Theory.

The fourth phase, **lexical and corpus phase**, is from 1990s to now. One is the increasing influence of lexicalized approach to grammar. The other is a revolution in natural language processing caused by the introduction of machine learning algorithms for language processing.

2.2 Processing Steps

There are several steps to process natural languages:

- **Morphological Processing:** Breaking chunks of language inputs into sets of tokens corresponding to paragraphs, sentences and words. E.g. sub-word tokens.
- **Syntax Parsing:** First, checking whether a sentence is well formed or not. For example, the sentence "The school goes to the boy" is not well formed and would be rejected by syntax parser. Second, breaking well formed sentence up into a structure that shows the syntactic relationships between different words.
- **Semantic Parsing:** First, checking for meaningfulness. For example, the sentence "Hot ice-cream" cannot be understood and would be rejected semantic parser. Second, drawing exact meaning from the text.
- **Pragmatic Analysis:** Fitting the actual objects/events, which is the given context obtained during Semantic Parsing. For example, the sentence "Put the banana in the basket on the shelf" can have two semantic interpretations and pragmatic Analyser will choose between these two possibilities.

2.3 Ambiguity

Ambiguity represents the capability of being understood in more than one way. Natural Language is ambiguous, there are several types of ambiguities:

- **Lexical Ambiguity** is the ambiguity of a single word. For example, the distinguish of a word as a noun, an adjective, or a verb.
- **Syntactic Ambiguity** occurs when a sentence could be parsed in different ways. For example, the sentence "The man saw the girl *with the telescope*" means whether the man saw the girl carrying a telescope or he saw her through his telescope?
- **Semantic Ambiguity** happens when a sentence contains an ambiguous word or phrase, which itself can be misinterpreted. For example, the sentence "The car hit the pole while *it* was moving" can be interpreted as either "The car, while moving, hit the pole" or "The car hit the pole while the pole was moving".
- **Anaphoric Ambiguity** arises due to the use of anaphora entities in discourse. For example, "the horse ran up the hill. *It* was very steep. *It* soon got tired." In which, the anaphoric reference of *It* in two situations cause ambiguity.
- **Pragmatic ambiguity** arises when the statement is not specific. For example, the sentence "I like you *too*" can be interpreted as "I like you just like you like me" or "I like you just like someone else does".

3 Definition

Natural Language Understanding (NLU) is a subtopic of NLP. It involves breaking down the human language into a machine-readable format and making use of grammatical rules and

common syntax to understand the meaning of text. NLU refers to how unstructured data is rearranged so that machines may "understand" and analyze it.

The application of NLU are automated reasoning, reading comprehension, machine translation, question answering, text categorization, voice-activation, archiving, and large-scale content analysis.

Task-Agnostic Problem in NLU

4 Tasks & Solutions

This section is to illustrate NLU's potential directions of improvements and relevant necessary concepts.

4.1 Word Level Analysis

Finite Automaton (FA) is an automaton having a finite number of states. Automaton is defined as an abstract self-propelled computing device that follows a predetermined sequence of operations automatically.

Deterministic Finite automation (DFA) is defined as the type of FA, where for every input we can determine the state to which the machine will move. DFA has a finite number of states.

Non-deterministic Finite Automation (NDFA) is defined as the type of FA, where for every input we cannot determine the state to which the machine will move and the machine can move to more than one combination of the states. NDFA has a finite number of states.

Regular Expression (RE) is a sequence of characters that define a search pattern. In NLP, RE helps to match/find other strings/sets of strings, using a specialized syntax held in a pattern. RE has been used to search text in UNIX and in MS WORD in identical way.

RE requires two things, one is the pattern that we wish to search, the other is a corpus of text from which we need to search. A RE can be defined as:

- ε is a RE and indicates that the language is having an empty string.
- φ is a RE and indicates that it is an empty language.
- If X and Y are REs, then their Concatenation, Union, and Kleen Closure are all REs.
- A string is a regular expression, if it is derived from above rules.

Regular Sets (RSs) represent the value of the RE. The properties of RSs are:

- If A is a RS, then the Complement, Reversal, and Closure of A are all RSs.
- If A and B are two RSs, then the Union, Intersection, Difference, and Concatenation of A and B are all RSs.

FA is the theoretical foundation of computation. REs is a way of describing FA. FA, RSs, and RGs are equivalent ways of describing regular languages.

Regular Grammar (RG) is a way to describe a regular language. It is a formal grammar that can be right-regular or left-regular.

Morphological Parsing is the task of recognizing and then breaking a word down into smaller morphemes and producing linguistic structure for the word. For example, the word *foxes* can be broken into two morphemes, *fox* and *es*. *fox* is a **stem**, a root of a given word. *es* is a **affix**, a grammatical function to the root of a given word. **Affix** has four types: Prefixes, Suffixes, Infixes, and Circumfixes. Both **stem** and **affix** are defined as **Morpheme**.

Word Order is decided by morphological parsing. The requirements for building a morphological parser are:

- **Lexicon** are the list of stems and affixes along with the **syntax** information.
- **Morphotactics** are the model of morpheme ordering of a word, which is to explain which classes of morphemes follow other classes of morphemes. E.g., the English plural morpheme always follows the noun rather than preceding it.
- **Orthographic rules** are used to model the changes in format while ordering a word. E.g., the rule of converting *y* to *ie* for *city*.

4.2 Syntactic Parsing

The purpose of Syntactic Parsing is first to check the text for meaningfulness based on the rules of formal grammar and then to draw exact meaning from the meaningful text.

Syntax is the set of rules, principles, and processes that govern the structure of sentence in a given language. One basic description of the syntax of a sentence is a sequence of **S**ubject, **V**erb, and **O**bject in a specific order. The variation of the sequence order in different languages lead to a different syntax structure. The difference is caused by a complex clausal phrase structure and is compatible with multiple derivations.

Parser is used to report and correct syntax error, create parse tree, create symbol table, and produce intermediate representations. **Top-down Parsing** constructs the parse tree from the start symbol and then transforms the start symbol to the input. **Bottom-up Parsing** starts with the input symbol and constructs the parser tree up to the start symbol.

Syntactic Parser is the task of generating constituency or dependency trees from a sentence depending upon the task for which inference is required. It is the process of analyzing the string of symbols in natural language conforming to the rules of formal grammar.

Derivation here is a set of syntactic parsing rules. During parsing, we need to decide the non-terminal state, which is replaced along with deciding the production rule. There are two types of derivations: **Left-most Derivation** is scanning and replacing the sentential form of an input from the left to right. **Right-most Derivation** is on the contrary.

Parse Tree is defined as the graphical depiction of a derivation. The start symbol of derivation is the root of the parse tree. The leaf nodes are terminals and interior nodes are non-terminals. In-order traversal will produce the original input string.

Grammar is essential to describe the syntactic structure of well-formed programs. A mathematical model of grammar was given by Noam Chomsky in 1956. Noam Chomsky introduced a **Constituency Grammar (CG)** based on the constituency relation, where all the related frameworks view the sentence structure in terms of constituency relation. The basic clause structure of it is understood in terms of noun phrase (NP) and verb phrase (VP).

Dependency Grammar (DG) introduced by Lucien Tesniere based on the dependency relation, which is opposite to CG. In DG, words are connected to each other by directed links.

The VP becomes the center of the clause structure. Every other syntactic units are connected to the VP in terms of directed links. These syntactic units are called dependencies.

Context Free Grammar (CFG) is a notation for describing languages and a superset of RG. CFG consists of a finite set of grammar rules containing a set of non-terminals (**V**), a set of terminals (**Σ**), a set of productions (**P**), and a start symbol (**S**).

4.3 Semantic Parsing

Semantic Parsing is the task of converting a natural language utterance to a logical form, so that a machine can understand the meaning of a natural language representation. It is a way of extracting meaning of a representation. The application of Semantic Parsing includes machine translation, question answering, and ontology induction.

Lexical Semantics is the first part of semantic parsing, which is the task of studying the meaning of individual words. Lexical items include morphemes, words, compound words, and phrases. Lexical semantics represents the relationship between lexical items. Semantic Parsing focuses on larger chunks, on the other hand, lexical analysis is based on smaller token.

The steps of Lexical Semantics are first, classifying of lexical items; second, decomposing of lexical items; and third, analyzing differences and similarities between various Lexical Semantic. Accordingly, Semantic Parsing has two steps: First, doing Lexical Semantics, and second, combining individual words to provide meaning of sentence.

The frame of semantic parsing has three steps: (1) decompose the sentence into lexical items; (2) cluster those items and label those clusters; (3) predict predicate relations between clusters. In which frames represent semantic representation of predicates, e.g. verbs. Clusters represent arguments.

Semantic Parsing is inherently more complicated than Syntactic Parsing, because Semantic Parsing is more about capturing the meaning of sentence rather than plain rule-based pattern matching.

Shallow Semantic Parsing is known as slot-filling or frame semantic parsing. It has a theoretical basis of frame semantics. Slot-filling systems are mechanisms for identifying the frame evoked by an utterance. Many architectures for slot-filling are variants of an encoder-decoder model, by encoding an utterance into a vector and decoding that vector into a sequence of slot labels.

Deep semantic parsing is known as compositional semantic parsing. It can parse arbitrary compositional utterances, by converting them to a formal meaning representation language. This is not doable by shallow semantic parsing. **Deep semantic parsing model** are either based on defining a formal grammar for a chart parser, i.e. Cornell Semantic Parsing Framework [2], Stanford University's Semantic Parsing with Execution (SEMPRE)[5], and the Word Alignment-based Semantic Parser (WASP)[86], or RNNs translating from a natural language to a meaning representation language.

The **datasets** used for training deep semantic parsing models are two main classes: One is used for question answering via knowledge base queries, i.e. Air Travel Information System (ATIS)[27] (standard dataset); GeoQuery[89] (benchmark dataset); and Overnight[84] testing how well semantic parsers adapt across multiple domains. The other is used for code generation, i.e. Constructed linking Hearthstone card texts to Python snippets[44]; IFTTT dataset[61] using a specialized domain-specific language with short conditional commands; Django dataset[58]

pairing Python snippets with English and Japanese pseudocode; and RoboCup dataset[37] pairing English rules with their representations in a domain-specific language that can be understood by virtual soccer-playing robots.

4.3.1 Models for Semantic Parsing

A list of language tasks:

- **Syntactic Parser**: Building constituency or dependency trees from a sentence.
- **Semantic Parser**: Mapping natural language text to formal representations.
- **Semantic Parsing**: Building a parse tree depending upon the specific given task.
- **Semantic Role Labeling (SRL)**: Given the structure of the target representation. The parsing would be done depending on a frame semantics.
- **Sequence Labeling Problem (SLP)** is to identify and label arguments (e.g., N, O, and V) in each sentence according to the given marked predicates for each sentence.
- **Formal Representation(FR) task** is similar to a Machine Translation (MT) problem translating between the natural and formal representations.

Solutions for specific task(s):

LSTM-based approach, i.e.*A Simple and Accurate Syntax-Agnostic Neural Model for Dependency-based Semantic Role Labeling* [48], takes advantage of the memory preservation property of LSTMs. Vectors are obtained from each word by concatenating pre-trained embeddings (Word2Vec), random embeddings, and randomly initialized POS embeddings. Predicate is (1-bit flag) marked by the word vector in a particular training instance. Word's context is obtained by fitting the word vector into a bi-LSTM layer. Words are labeled based on a softmax classifier obtained by the dot product of its hidden state with the predicate's hidden state. Weight matrix parameterized on the row labeled r.

Graph Convolutional Network (GCN)-based approach, i.e.,*Graph Convolutional Networks for Text Classification*[87], has been used to represent the dependency tree for the sentence. Which means that a GCN input layer encodes the sentence into an $m \times n$ matrix based on its dependency tree, such that each of the n nodes of the tree is represented as an $m \times 1$ vector. Once such a matrix has been obtained, we can perform convolutions on it.

One weakness of a one-layer GCN is that it can only capture information about its immediate neighbor. However, this could be solved by stacking GCN layers, so that one can incorporate higher degree neighborhoods.

GCNs and LSTMs are complementary. LSTMs capture long-term dependencies well but are not able to represent syntax effectively. On the other hand, GCNs are built directly on top of a syntactic-dependency tree so they capture syntax well, but one can only capture information about its immediate neighbor. Therefore, using a GCN layer on top of the hidden states obtained from a bi-LSTM layer would theoretically capture the best of both worlds. This hypothesis has also been corroborated through experimental results.

Encoder-decoder model, i.e.*Language to Logical Form with Neural Attention*[16], can solve both FR and SLP. An encoder converts the input sequence to a vector representation and a

decoder obtains the target sequence from this vector. The encoder uses a bi-LSTM layer to obtain the vector representation of the input sequence. The final hidden state is fed into the decoder layer, which is again a bi-LSTM. The hidden states obtained from this layer is used to predict the corresponding output tokens using a softmax function. Alternatively, we can have a hierarchical decoder to account for the hierarchical structure of logical forms. For this purpose, we introduce a non-terminal token to indicate the start of a sub-tree. To incorporate the tree structure, we concatenate the hidden state of the parent non-terminal with every child. Finally in the decoding step, using an attention layer where the context vector is a weighted sum over the hidden vectors in the encoder to better utilize relevant information from the input sequence.

LSTM-based approach[48], GCN-based approach[87], and Encoder-decoder model[16] are supervised approaches for SLP. Encoder-decoder model[16] is also for FR. However, those are constrained by cost and availability of annotated data, especially since manually labeling semantic parsing is a time-consuming process. Recent years there is a transition from using statistical methods to generative models for NLP tasks, so next we would introduce some unsupervised or semi-supervised approaches to address this constrains.

Generative models, i.e.*A Bayesian Model for Unsupervised Semantic Parsing*[72], which makes use of statistical processes to model semantic parsing. Basically we get a semantic frame from the PY process, and then generate the corresponding syntax from a Dirichlet process. This is done recursively. For the root level parameters, a stick-breaking construction is used. Below is the essence of the generative algorithm of Bayesian model:

- Obtain the semantic class for the root of the tree from the probability distribution, which is a sample drawn from the distribution of semantic classes given by a **hierarchical Pitman-Yor (PY) process**.
- Once the root is obtained, we call the function **GenSemClass** on this root.
- Since the current root only has a semantic class, we obtain its syntactic realization from a distribution over all possible syntactic realizations, which is given as a **Dirichlet Process** with the arguments as the base word and a prior. Essentially, the base word x is obtained from a geometric distribution, and the subsequent words are obtained by computing the conditional probability $P(y|x)$, and the next word $P(z|y)$.
- For each argument type t, if the probability of having at least 1 argument of type t is non-zero, we generate an argument of that type using function **GenArgument**, until that probability becomes 0. The GenArgument function again computes the base argument from the distribution of syntactic realizations, and then obtains the next semantic class again from the hierarchical PY process.
- We then recursively call the GenSemClass function on this new class.

Next we would introduce some approaches in a neural framework for semantic parsing, i.e. Transfer Learning for Neural Semantic Parsing[19] and Deep Multitask Learning for Semantic Dependency Parsing[60].

Transfer Learning for Neural Semantic Parsing[19] uses sequence-to-sequence model developed for MT. The sentence is first encoded into an intermediate vector representation and then decoded into an embedding representation for the parse tree. Popular encoders and decoders are stacked bidirectional LSTM layers with some attention mechanism. While reading the output embedding at each step, the model has 2 options:

A COPY-WRITE mechanism: the model has 2 options while reading the output embedding at each step:

- COPY: This copies one symbol from the input to the output.
- WRITE: This selects one symbol from the vocabulary of all possible outputs.

A final softmax layer generates a probability distribution over both of these choices: the probability of choosing WRITE at any step is proportional to an exponential over the output vector at that step; and the probability of choosing COPY is proportional to an exponential over a non-linear function of the intermediate representation and the output vector, i.e., the encoded and decoded vectors.

Furthermore there are 3 ways to extend this transfer learning method to a multi-task setting:

- One-to-many: One shared encoder; Each task has its own decoder and attention parameters.
- One-to-one: One shared entire sequence with an added token at the beginning to identify the task.
- One-to-shareMany: This also has a shared encoder and decoder, but the final layer is independent for each task. A large number of parameters can be shared among tasks while still keeping them sufficiently distinct. Empirically, this model was found to perform best among the three.

Deep Multitask Learning for Semantic Dependency Parsing[60] Given sentence x and a set of all possible semantic graphs for that sentence $Y(x)$, we want to compute: the scoring function S is a sum of local scores, each of which is itself a parameterized function of some local feature - first order logic: Predicate, Unlabeled arc, and Labeled arc.

For the 2 input words, we first obtain vectors using a bi-LSTM layer, and these are then fed into multi-layer perceptrons (MLPs) corresponding to each of the three local features. Each first-order structure is itself associated with a vector (shown in red). The scoring function $s(p)$ is simply the dot product of the MLPs output and the first-order vector. The cost function is a max-margin objective with a regularization parameter and a sum over individual losses.

Once this basic architecture is in place, there are two methods to extend it with transfer learning. The task here are three different forms in semantic dependency parsing including Delph-in MRS, Predicate-Argument Structure, and Prague Semantic Dependencies, so that each of these require a different variation of the output form. In the first method, the representation is shared among all tasks but the scoring is done separately. This further has variants wherein we can either have a single common bi-LSTM for all tasks or a concatenation of independent and common layers. The second method describes a joint technique to perform representation and inference learning across all the tasks simultaneously. The description is mathematically involved but intuitively simple, since we are just expressing the inner product in the scoring function in a higher dimension.

4.4 Meaning Representation

Semantic Parsing creates a representation of the meaning of a sentence. The following components of Semantic System play an important role in **word representation**.

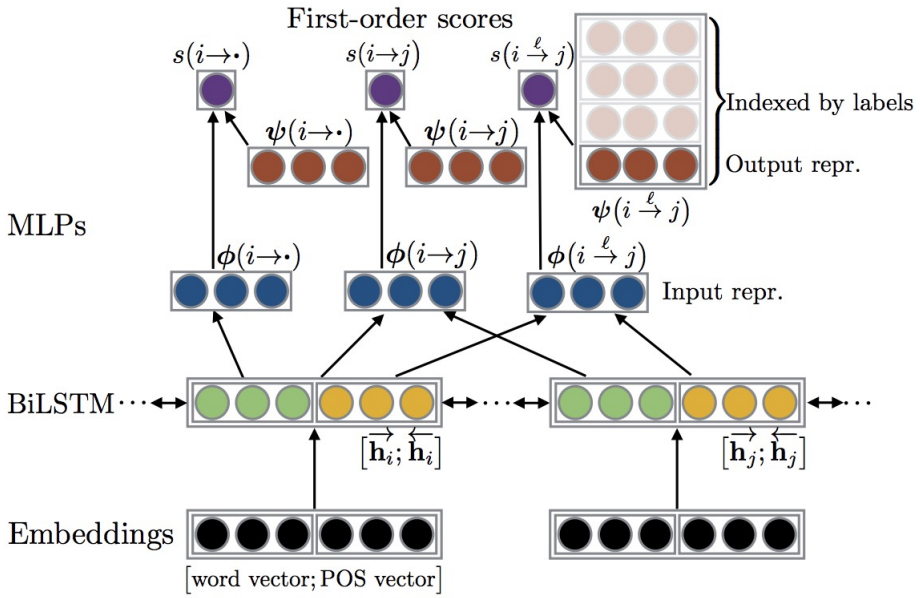


Figure 1: Illustration of the architecture of the basic model: Deep Multitask Learning for Semantic Dependency Parsing. i and j denote the indices of tokens in the given sentence. The figure depicts single-layer BiLSTM and MLPs, while in practice we use two layers for both [60]

- **Entities** represent particular individuals. E.g., Haryana, India, and Ram.
- **Concepts** represent the general category of individuals. E.g., animal and person.
- **Relations** represent the relationship between entities and concepts. E.g., Ram is a person.
- **Predicates** represent the verb structures. E.g., semantic roles and case grammar.

4.4.1 Approaches for Meaning Representations

Some approaches that Semantic Parsing uses for **meaning representations** are First order predicate logic (FOPL), Semantic Nets, Frames, Conceptual Dependency (CD), Rule-based architecture, Case Grammar, and Conceptual Graphs. The reasons of meaning representations are that we need to link the linguistic elements to non-linguistic elements, make representation variety at lexical level, and then use it for reasoning.

4.5 Word Sense Disambiguation

Lexical, syntactic, or semantic ambiguity, is one of the very first problem that any NLP system faces. Part-of-speech (POS) taggers with high level of accuracy can solve Word's syntactic ambiguity.

Word sense disambiguation (WSD) means resolving semantic ambiguity, which is harder to be solved than syntactic ambiguity. An example of WSD is

- I can hear *bass* sound. (*bass* means *frequency*)
- He likes to eat grilled *bass*. (*bass* means *fish*)

4.5.1 Methods to Word Sense Disambiguation

Knowledge-based Methods rely on dictionaries, treasures, and lexical knowledge base, instead of corpora evidences. The **Lesk method** is the seminal dictionary-based method, which was introduced by Micheal Lesk in 1986. In 2000, Kilgarriff and Rosensweig further simplified it as "measure overlap between sense definitions of word and the set of words in surrounding sentence or paragraph", which means identifying the correct sense for one word at a time.

Supervised learning Methods is a machine learning method using sense-annotated corpora to train. The context is represented as a set of "features" of the words including the information about the surrounding words also. Assume the context can provide enough evidence to disambiguate the sense, so that the words knowledge and reasoning are unnecessary. The most successful approaches are **support vector machine** and **memory-based learning**, those rely on substantial amount of and expensive manually sense-tagged corpora.

Semi-supervised learning Methods are widely used, since the lack of training corpus, i.e.**bootstrapping from seed data**. It uses very small amount of annotated text(labeled data) and large amount of plain unannotated text(unlabeled data).

Unsupervised learning Methods are used to clustering similar context, i.e., word sense induction and word sense discrimination. For example clustering similar context based on word occurrences. This methods assume that similar senses occur in similar contexts. Unsupervised learning have great potential to overcome the knowledge acquisition bottleneck due to non-dependency on manual efforts.

Evaluation

To evaluate the WSD, two inputs are required. One is **dictionary**, which is used to specify the senses to be disambiguated. The other is **test corpus**, which has the target correct senses, with two possible types. One is **lexical sample** for small sample of words, the other is **all-words** for a piece of text.

Application

WSD is widely applied in almost every application of language technology. **Machine Translation (MT)** is the most obvious application of WSD. Lexical choice for the words that have distinct translations for different senses, is done by WSD. The senses in MT are represented as words in the target language. Most of the MT systems do not use explicit WSD module. As like MT, current **Information Retrieval (IR)** systems do not explicitly use WSD module. They rely on the concept that user would type enough context in the query to only retrieve relevant documents. In **Information Extraction (IE)** WSD is necessary to do accurate analysis of text. For example, WSD helps medical intelligent system to flag "illegal drugs" rather than "medical drugs". In addition, WSD and **lexicography** can work together in loop since WSD is a supplement of lexicography. WSD provides rough empirical sense groupings and statistically significant contextual indicators of sense, on the other hand, modern lexicography is corpus-based.

Difficulties

There are four main difficulties in WSD. First, different senses can be very closely related, so that it is hard to decide the sense. Second, completely different algorithm is needed for different applications, i.e. MT takes the form of target word selection and IR does not require a form of target word. Third, words cannot be easily divided into discrete sub-meanings. Fourth, inter-judge variance problem, which means WSD systems are generally tested by having their

results on a task compared against the task of human beings.

5 Core Language Models

5.1 Bayesian Theorem

This section is a review of previous work and background information of Bayes Theorem.

Disjoint: For every (possibly infinite) collection of disjoint sets $A_k, k = 1, 2, 3...$, we have

$$\mathbb{P}[\cup_{k\geqslant 1} A_k] = \sum_{k\geqslant 1} \mathbb{P}[A_k].$$

Independent events: Events happening (or not happening) does not depend on one another, we have

$$\mathbb{P}(E_1 \cap E_2) = \mathbb{P}(E_1)\mathbb{P}(E_2).$$

Conditional Probability: For any two events A and B with $\mathbb{P}(B) > 0$, the conditional probability of A given that B has occurred is:

$$\mathbb{P}[A|B] = \frac{\mathbb{P}[A \cap B]}{\mathbb{P}[B]}.$$

Bayes Theorem: Let A_1, A_2,..., A_k, be k mutually exclusive and exhaustive events with $\mathbb{P}[A_i] > 0$ for $i = 1, 2, ..., k$. Then for any other event B for which $\mathbb{P}[B] > 0$,

$$\mathbb{P}[A_j|B] = \frac{\mathbb{P}[B|A_j]\mathbb{P}[A_j]}{\sum_i \mathbb{P}[B|A_i]\mathbb{P}[A_i]}.$$

Bayes Theorem example:Conditional on θ, $X \sim Poisson(\theta)$, for $k = 0, 1, 2, 3...$,

$$\mathbb{P}[X = k|\theta] = e^{-\theta}\frac{\theta^k}{k!}.$$

Bayesian statistical principles: likelihood, prior, posterior.

Given the **likelihood**, i.e. the pdf/pmf of X given θ: $[X|\theta] \sim f(x;\theta)$, for $x \in X, \theta \in \Theta$.

Given the pdf/pmf **prior** on the parameter space: $[\theta] \sim h(\theta)$, for $\theta \in \Theta$.

Then based on Bayes Theorem, the pdf/pmf **posterior**: $[\theta|X] = \frac{f(x;\theta)h(\theta)}{\int_\theta f(x;\theta)h(\theta)d\theta}$.

Proof:

Joint pdf/pmf of (X, θ): $f(x;\theta)h(\theta)$.

Marginal pdf/pmf of X: $m(x) = \int_\theta f(x;\theta)h(\theta)d\theta$

Thus, the posterior:

$$\begin{aligned}[\theta|X] &= \frac{f(x;\theta)h(\theta)}{\int_\theta f(x;\theta)h(\theta)d\theta} \\ &= \frac{f(x;\theta)h(\theta)}{m(x)} \\ &\propto f(x;\theta)h(\theta).\end{aligned}$$

Notice:

- The posterior distribution $[\theta|X]$ leads to many AII/ML/Statistical techniques.
- The posterior distribution can be used for **point estimation**, **interval estimation**, **hypothesis testing**, and much more.
- The posterior $[\theta|X]$ is very challenging to obtain.
- The choice of the prior is critical, especially in small samples or for high dimensional parameters.

The prior is a **conjugate prior**, if the prior and the posterior are the same brand of pdf/pmf distributions (but with different parameters).

conjugate prior example: Given $[X|\theta] \sim \text{Binomial}(n,\theta)$ and $[X] \sim \text{Beta}(\alpha,\beta)$, then $[X|\theta] \sim \text{Beta}(\alpha+X, \beta+n-X)$. *Proof*:

$$\begin{aligned}[X|\theta] &\propto \left\{\binom{n}{X}\theta^X(1-\theta)^{n-X}\right\}\left\{\frac{\lceil(\alpha+\beta)}{\lceil(\alpha)+\lceil(\beta)}\theta^{\alpha-1}(1-\theta)^{\beta-1}\right\} \\ &\propto \theta^{\alpha+X-1}(1-\theta)^{\beta+n-X-1}\end{aligned}$$

Therefore, this can now be seen ass a $\text{Beta}(\alpha+X, \beta+n-X)$ distribution.

Bayesian point estimators are obtained by minimizing $\int_\theta L(\theta, T(X))[\theta;X]d\theta$, for some loss function $L(\theta, T(X))$. All Bayes estimators are functions of the posterior distribution.

Posterior mean: $\int_\theta \theta[\theta;X]d\theta$, is the Bayes estimate when $L(\theta,T(X)) = (\theta - T(X))^2$.

Posterior median: $\int_\theta \theta[\theta;X]d\theta$, is the Bayes estimate when $L(\theta,T(X)) = |\theta - T(X)|$.

MAP estimator: $\int_\theta \theta[\theta;X]d\theta$, is the point where the posterior is maximized (very similar to MLE). This corresponds to the mode of the posterior distribution.

Bayesian interval estimation are comparable to confidence intervals for non-Bayesian statistics. There are intervals that contains θ with apre-specified probability.

- HDR/HPD regions: Regions with high posterior pdf/pmf. (These can be hard to obtain if the posterior pdf is not unimodal.)
- Credible interval: An interval having the correct coverage. (Usually, we want short credible intervals.)

5.1.1 Bayesian Deep Learning

Bayesian deep learning[81] is a probabilistic framework to unify modern deep learning and probabilistic graphical models.

Bayesian deep learning in a broader sense (BDL) is the combination of a probabilistic neural network and a probabilistic graphical model. This combination enables end-to-end learning and inference. Bayesian deep learning in a narrower sense is the Bayesian version of probabilistic neural networks (BNNs), which is a component of BDL. (In this report, the term BDL refers to BDL in a broader sense.)

The advantage of BDL compared with traditional deep neural networks is that BDL including both deep neural networks and probabilistic graphical models, so that it provide a unified deep learning framework that can supports all four functionalities listed below:

- conditional inference;
- causal inference;
- logic deduction; and
- uncertainty modeling.

However, a traditional deep neural network, even though borrow some ideas from probabilistic graphical models, one can only enable a subset of the four functionalities.

BNNs is proposed by David MacKay in 1992 [46]. In 2014, Collaborative Deep Learning (CDL) [79], a concrete application of the BDL framework to recommender systems, significantly improved recommender systems' performance. In 2015, CDL has been generalized into a BDL framework [80]. So far a diverse and a large amount of models across various domains are proposed under this BDL framework. Table 1 is a table of selected variations under the framework of BDL[81], which is useful for the improvement of systems' NLU abilities.

To better understand the internal structure of BDL, we need to be familiar with the key concepts of BDL: one framework, two components, and three variable sets.

- One framework: The BDL general framework.
- Two components:

  Perception components: probabilistic neural networks, eg. Restricted Boltzmann Machine (RBM), Probabilistic Autoencoder [79], VAE [34], and Natural-Parameter Network [77].

  Task-specific components: traditional (static) Bayesian networks, deep Bayesian networks [82], stochastic processes [30], and dynamic Bayesian network[55].

- Three variable sets:

  Perception variables: variables inside the **perception component**. These variables are drawn from relatively simple distributions (e.g., Dirac delta distributions or Gaussian distributions), and the graph among them is usually simple as well. This is to ensure low computational complexity; otherwise multiple layers of **perception variables** will be computationally prohibitive. This is actually the reason why we need probabilistic neural networks here, since they can be efficiently learned via backpropagation (BP).

  Hinge variables: variables inside the **task-specific component** with direct connections to the **perception component**. Their job is to connect these two components and facilitate bidirectional, quickly communicate between them.

Table 1: Summary of BDL Models with Different Variance Types (ZV: Zero-Variance, HV: Hyper-Variance, LV: Learnable-Variance)

Application	Models	Variance
Recommender Systems	Collaborative Deep Learning(CDL) [79]	HV
	Bayesian CDL [79]	HV
	Marginalized CDL [40]	LV
	Symmetric CDL [40]	LV
	Collaborative Deep Ranking [88]	HV
	Collaborative Knowledge Base Embedding [90]	HV
	Collaborative Recurrent AE [76]	HV
	Collaborative Variational Autoencoders [41]	HV
Topic Models	Relational SDAE	HV
	Deep Poisson Factor Analysis with Sigmoid Belief Networks [22]	ZV
	Deep Poisson Factor Analysis with Restricted Boltzmann Machine [22]	ZV
	Deep Latent Dirichlet Allocation [12]	LV
	Dirichlet Belief Networks [92]	LV
NLP	Sequence to Better Sequence [53]	LV
	Quantifiable Sequence Editing [43]	LV
Link Prediction	Relational Deep Learning [78]	LV
	Graphite [24]	LV
	Deep Generative Latent Feature Relational Model[49]	LV

Zero-Variance **(ZV)**: Assume no uncertainty during the information exchange between the two components.
Hyper-Variance **(HV)**: Assume that uncertainty during the information exchange is defined through hyperparameters.
Learnable Variance **(LV)**: Using learnable parameters to represent uncertainty during the information exchange.

Task variables: variables inside the **task-specific component** without direct connections to the **perception component**. In contrast to **perception variables**, **task variables** can be drawn from various complex distributions and the graph connecting them can be more complicated. This is to better describe the complex conditional dependencies among the task variables.

5.2 Topic Modeling

Topic modeling is to extract thematic information from large corpus of documents. A long line of work on topic modeling sets up the basic structure for topic models, starting from probabilistic latent semantic indexing (pLSI), followed by Latent Dirichlet Allocation(LDA), Correlated Topic Models(CTM), Pachinko allocation and many others.

According to these works, each topic is a probability distribution over words and each document is a probability distribution over topics. These models also has the "bag-of-words" assumptions, which means that the ordering of the words is not crucial in determining the topics. Hence words in a document are generated independently at random. To generate a word, first generate its topic according to the document-topic distribution, then pick a word from the corresponding topic-word distribution.

The process of generation a word can be summarized as a matrix product. The latent variable are the document-topic distributions. The linear transform is constructed from the topic-word distributions. That is, the (i,j)-th entry of matrix $W \in \mathbb{R}^{r\times m}$ corresponds to the probability that document j generates topic i. The (i,j)-th entry of $A \in \mathbb{R}^{n\times r}$ corresponds to the probability that the topic i generates the word l. The (i,j)-th entry of product AW is exactly the probability that the topic i generates the word l. The observation function f samples N words according to the distribution AW_j.

Therefore, we have the following description of topic modeling in General Matrix Factorization (GMF) and Non-negative Matrix Factorization (NMF) frameworks.

GMF framework

Given: an unknown topic matrix A with non-negative entries that is dimension $n \times r$, and a stochastic generated unknown matrix W that is dimension $r \times m$. Each column of AW is viewed as a probability distribution on rows, and for each column we are given $N \ll n$ i.i.d samples from the associated distribution.

Goal: Reconstruct A and parameters of the generating distribution for W.

Directly analyzing the GMF problem for topic modeling is not easy because much information is lost in the sampling process.

NMF framework

Given: Matrix $M = AW + noise$, where matrices $A \in \mathbb{R}^{n\times r}$ and $W \in \mathbb{R}^{r\times m}$ have non-negative entries.

Goal: Find A, W.

A natural assumption called "separability" simplifies the problem, and gives a polynomial time algorithm when the instance is separable.

Separability Assumption: A non-negative factorization $M = AW$ is separable if for each column i of A, there is some row $r(i)$ of A that has a single nonzero entry and this entry is in the i-th column. Under this assumption, the NMF problem has a nice geometric interpretation that leads to polynomial time algorithms.

For topic modeling the separability assumption naturally translates to the "anchor words assumption". **Anchor words assumption:** Every topic has a word with probability at least p in that topic, and has probability 0 in all other topics. This assumption greatly simplifies the learning task. Even though we get samples from AW which from a very coarse approximation of the product, we show the noise can be reduced.

However, the disadvantage of the above algorithm is: (1) the algorithm is very slow in practice, because it requires solving many linear programs. (2) the algorithm is unstable and produce negative entries, because the recovery algorithm relies on matrix inversion.

To address those problems in the given two new algorithm bellow, we (1) replace matrix inversion with a new gradient-based inference method by presenting a simple probabilistic interpretation

of topic recovery given anchor words. (2) the new algorithm produces results run orders of magnitude faster and is comparable to the best MCMC implementations.

Algorithm 1 High Level Algorithm

Input: Textual corpus D, Number of anchors K, Tolerance parameters ϵ_a $\epsilon_b > 0$.
Output: Word-topic matrix A, topic-topic matrix R.

$Q \leftarrow$ Word Co-occurences (D)
From $\{\overline{\mathrm{Q_1}}, \overline{\mathrm{Q_2}}, ...\overline{\mathrm{Q_V}}\}$, the normalized rows of Q.
$\mathbf{S} \leftarrow$ FastAnchorWords($\{\overline{\mathrm{Q_1}}, \overline{\mathrm{Q_2}}, ...\overline{\mathrm{Q_V}}\}$, K, ϵ_a)
$A, R \leftarrow$ RecoverKL($Q, \mathbf{S}, \epsilon_b$) (Algorithm 2)
return A, R

Algorithm 2 RecoverKL

Input: Matrix Q, Set of anchor words $\mathbf{S}$, Tolerance parameters ϵ.
Output: Matrix A, R

Normalize the rows of Q to form $\overline{\mathrm{Q}}$
Store the normalization constants $\mathbf{p}_w = Q\mathbf{1}$
$\overline{\mathrm{Q_{s_k}}}$ is the row of $\overline{\mathrm{Q}}$ for the k^{th} anchor word
for `i = 1,..., V` **do**
    Solve $C_i = \arg\, min_{C_i}\ D_{KL}(\overline{\mathrm{Q_i}} \,\|\sum k \in \mathbf{S} C_{i,k} \overline{\mathrm{Q_{s_k}}}\ )$
    Subject to: $\sum k C_{i,k} = 1$ and $C_{i,k} \geqslant 0$
    With tolerance: ϵ
$A^{'} = diag(\mathbf{p}_w)C$
Normalize the columns of $A^{'}$ to form A.
$R = A^{*}\ Q\ A^{*T}$
return A, R

5.3 Multi-Task learning & Meta-Learning

Standard computer vision used hand-designed features, e.g., HOG, DPM, and SVM. Modern computer vision used end-to-end training. Deep learning allows us to handle unstructured inputs, e.g., pixels, language, sensor readings, etc., without hand-engineering features, with less domain knowledge.

In deep learning for object classification, the AlexNet reduced the error rate to less than 0.2 in 2012, which is a great improvement. Also from then on, other deep learning models graduate achieved or exceeded the state-of-the-art score. Deep learning for machine translation also transformed from Phrase-based machine translation (PBMT) to Google's neural machine translation (GNMT) in 2016.

However, those models are environmental-oriented or task-oriented. That is to say that we cannot use a well trained model to solve for newly emergent task.

So deep learning technologies have been used to train on large and diverse data to generate large models in order to achieve broad generalization, e.g., GPT-2, Transformer, imagenet, etc. However, if we don't have a large dataset, for example, we don't have enough data for task-agnostic problem, then it is impractical to learn from scratch, e.g., for each disease, each language, each person, etc.

Another thing is the data structure, what if we have a dataset which has a long tail instead of normal distribution or Gaussian distribution? This case could happen in the case that objects encountered interactions with people.

To quickly learn something new, we could leverage prior experience using few-shot learning.

So in order to solve for problems illustrated above including a general-purpose AI system,

encountering data limitations, unforeseen data structures, and quickly learn something new, multi-task learning is one way to solve.

Task Definition:

Dataset D and Lass function $L \rightarrow$ model f_θ.

Different tasks can vary based on: different objects, different people, different objectives, different lighting conditions, different words, different languages, etc.

Problem Definition:

Multi-task learning problem: Learn all of the tasks more quickly or more proficiently than learning them independently.

Meta-Learning problem: Given data/experience on previous tasks, learn a new task quickly and/or more proficiently.

Domain adaptation: In some ways it is a form of transfer learning. Assume the task domain might be out of distribution from what you are seeing during training.

Doesn't a multi-task learning reduce to single-task learning?

$$D = \bigcup D_i, L = \bigcup L_i$$

Yes, but we can do better since we know that they come from different domain of tasks. The application of multi-task learning are multilingual machine translation, one-shot imitation learning from humans, multi-domain learning for sim2real transfer, YouTube recommendation (multi-task and multi-objective systems developing algorithms that can handle multiple competing objectives).

5.3.1 Multi-Task Learning

Models training

A general deterministic function: $f_\theta(y|x)$, where θ represents the weight, the neural network producing a distribution over output y given the input x.

Single-task learning(supervised): $\mathscr{D} = \{(x, y)_k\}$, a dataset of many input and output pairs.

Typical loss: negative log likelihood: $\mathscr{L}(\theta, \mathscr{D}) = -\mathbb{E}_{(x,y)\sim\mathscr{D}}[log f_\theta(y|x)]$. We would like to minimize the loss functions $min_\theta \mathscr{L}(\theta, \mathscr{D})$ with respect to the parameters.

A task: $\mathscr{T} \triangleq \{p_i(x), p_i(y|x), \mathscr{L}_i\}$, where $p_i(x)$ is the distribution over the inputs, $P_i(y|x)$ is the distribution over the labels given the inputs and a loss function. So these two distribution p are correspond to the true data generating distributions, where we do not access to.

Corresponding datasets: $\mathscr{D}_i$ (as shorthand for $\mathscr{D}_i^{train}$) and $\mathscr{D}_i^{test}$.

Multi-task classification: $\mathscr{L}_i$ same across all tasks. e.g. the cross entropy loss. But the input may vary across different tasks. e.g. per-language handwriting recognition, personalized spam filter.

Multi-label learning: $\mathscr{L}_i$ and $p_i(X)$ are same across all tasks. The task may make predictions for different labels. e.g. CelebA attribute recognition, scene understanding($L_{tot} = w_{depth}L_{depth} + w_{kpt}L_{kpt} + w_{normals}L_{normals}$).

$\mathscr{L}_i$ **may vary across tasks.** There are two settings this would happen. One setting maybe one

task corresponds to predicting a discrete variable, whereas another one corresponds to predicting a continuous variable. So that one might correspond to a cross-entropy loss function, whereas another might correspond to mean squared error. Another setting is when we care more about one task than another task. So that we may have a loss function weight corresponding to one task that is higher than the weight of another task.

Sometimes we have $f_\theta(y|x, z_i)$ e.g., one-hot encoding of the task index or, meta-data

- personalization: user features/attributes
- language description of the task
- formal specifications of the task

Objective: $min_\theta \sum_{i=1}^{T} \mathscr{L}_i(\theta, \mathscr{D}_i)$

A model decision and an algorithm decision: How should we condition on z_i? How to optimize objective?

Conditioning on the task Let's assume z_i is the task index. Question: How to condition on z_i in neural network models in order to share as little as possible between different tasks?

Answer: To have separate networks for each tasks, each of them have completely separate weights. Each tasks corresponding to multiplicative gating $y = \sum_j 1(z_i = j)y_j$, so that each tasks independently trained within a single network with no shared parameters (weights) across tasks.

The other extreme Concatenate z_i with input and/or activation, all parameters are shared except the parameters directly following z_i.

An alternative view on the Multi-Task Objective:

Split *theta* into shared parameters θ^{sh} and task-specific parameters θ^i.

Then, our objective is: $min_{\theta^{sh},\theta^1,\ldots,\theta^T} \sum_{i=1}^{T} \mathscr{L}_i(\{\theta^{sh}, \theta^i\}, \mathscr{D}_i)$

Choosing how to condition on z_i equivalent to choosing how where to share parameters.

Conditioning: Some Common Choices

- **Concatenation-based** conditioning concatenates the conditioning representation to the input. The result is passed through a linear layer to produce the output.
- **Conditional biasing** first maps the conditioning representation to a bias vector. The bias vector is then added to the input.
- **Multi-head architecture**. Input first passes through the shared layers and then passes through the task-specific layers.
- **Multiplicative** conditioning. Conditional scaling first maps the conditioning representation to a scaling vector.

Concatenation-based conditioning and Conditional biasing can work together. Why might multiplicative conditioning be a good idea? First, it is more expressive than additive conditioning. Second, multiplicative gating is more naturally represented by multiplicative conditioning. similarly, to choose a specific part of the network for different tasks, multiplicative conditioning

performed better than multi-head architecture. Multiplicative conditioning generalizes independent networks and independent heads.

Conditioning: More Complex Choices More complex architecture include various modules, attention, components, and different gating mechanisms. For example, Cross-Stitch Networks, Multi-Task Attention Network, Deep Relation Networks, and Sluice Networks.

However, these design decisions are like neural network architecture tuning: problem dependent, largely guided by intuition or knowledge of the problem, currently more of an art than a science.

To **Optimizing the objective** $min_\theta \sum_{i=1}^{T} \mathscr{L}_i(\theta, \mathscr{D}_i)$, basic version are:

- Sample mini-batch of tasks $\mathscr{B} \sim \{\mathscr{T}_i\}$
- Sample mini-batch data points for each tasks $\mathscr{D}_i^b \sim \mathscr{D}_i$
- Compute loss on the mini-batch: $\hat{\mathscr{L}}(\theta, \mathscr{B}) = \sum_{\mathscr{T}_k \in \mathscr{B}} \mathscr{L}_k(\theta, \mathscr{D}_k^b)$
- Back propagate loss to compute gradient $\nabla_\theta \hat{\mathscr{L}}$
- Apply gradient with neural net optimizer (e.g. Aham)

Note: This ensures that tasks are sampled uniformly, regardless of data quantities.

Tip: for regression problems, make sure your task labels are on the same scale!

Challenges

- Challenge1: Negative transfer. Solution: sometimes independent networks work the best. Reasons1: optimization challenges - caused by cross-task interference; tasks may learn at different rates. Reason2: limited representational capacity - multi-task networks often need to be much large than their single-task counterparts. Alternative Solution1: If having negative transfer, share less cross tasks: binary decision plus "Soft parameter sharing":

  $$min_{\theta^{sh}, \theta^1, \ldots, \theta^T} \sum_{i=1}^{T} \mathscr{L}_i(\{\theta^{sh}, \theta^i\}, \mathscr{D}_i) + \sum_{i'=1}^{T} ||\theta^t - \theta^{t'}||.$$

  The benefit of it is that it allows for more fluid degrees of parameter sharing. But the downside is that when thinking about algorithm, we need to tune yet another set of design decision/hyper parameters.

- Challenge2: Overfitting. Reason1: we may not share enough parameters. Solution: sharing more, so that Multi-task learning would be equivalent to a form of regularization.

Case study of real-word multi-task learning

Recommending What Video to Watch Next: A Multitask Ranking System Goal: Make recommendations for YouTube. Conflicting objectives: videos that users will rate highly; videos that users will share; videos that users will watch.

Implicit bias caused by feedback - user may have watched it because it was recommended.

Framework of the case

The Ranking Problem

The Architecture

Problem

Solution

The Architecture Set-up

Results

5.3.2 Meta-Learning

There are two ways to view meta-learning algorithms: Mechanistic view and Probabilistic view.

Mechanistic view:

- Deep neural network model that can read in an entire dataset and make predictions for new data points.
- Training this network uses a meta-dataset(a dataset of datasets, each for a different task), which itself consists of many datasets, each for a different task.
- This view makes it easier to implement meta-learning algorithms.

Probabilistic view:

- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks.
- Learning a new task uses this prior and (small) training set to infer most likely posterior.
- This view makes it easier to understand meta-learning algorithms

Problem Definition

Supervised learning:

$$\arg\max_{\phi} log(\phi|D) \tag{1}$$

$$= \arg\max_{\phi} log(D|\phi) + logp(\phi) \tag{2}$$

$$= \arg\max_{\phi} \sum_{i} logp(y_i|x_i, \phi) + logp(\phi). \tag{3}$$

$D = \{(x_1, y_1), ..., (x_k, y_k)\}$, where input $x_1, ..., x_k$ and $y_1, ..., y_k$ are labels, D is training data, ϕ is model parameters, and $p(D|\phi)$ is data likelihood, $logp(\phi)$ is weight decay.

What is wrong with this? The most powerful models typically require large amounts of labeled data. Labeled data for some tasks may be very limited.

The Meta-Learning Problem

$\arg\max_{\phi} log(\phi|D) D = \{(x_1, y_1), ..., (x_k, y_k)\}$

$\arg\max_{\phi} log(\phi \mid D)$	$D = \{(x_1, y_1), ..., (x_k, y_k)\}$
can we incorporate additional data?	$D_{meta-train} = \{D_1, ..., D_n\}$
$\arg\max_{\phi} log(\phi \mid D, D_{meta-train})$	$D_i = \{(x_1^i, y_1^i), ..., (x_k^i, y_k^i)\}$

What if we do not want to keep $D_{meta-train}$ around forever?

Meta-learning can bee viewed as a process of learning a set of meta-parameters θ: $p(\theta|D_{meta-train})$, where θ summarizes meta-training data such that thee problem could be solved quickly. The dataset is defined as $D_{meta-train} = \{(D_1^{tr}, D_1^{ts}), ..., (D_n^{tr}, D_n^{ts})\}$, where $D_i^{tr} = \{(x_1^i, y_1^i), ..., (x_k^i, y_k^i)\}$ and $D_i^{ts} = \{(x_1^i, y_1^i), ..., (x_l^i, y_l^i)\}$. So meta-learning is trying to maximize the likelihood of the set of meta-parameters: $\theta^* = \arg\max_\theta \, log\, p(\theta|D_{meta-train})$.

The adaptation process is adapting those parameters to compute the set of parameters ϕ that can solve a new task given a training dataset for that task and the meta-parameters that we just learned above. So the adaptation function is defined as: $\phi^* = \arg\max_\phi log\, p(\phi|D^{tr}, \theta^*)$. For short, this function could be written as: $\phi^* = f_{\theta*}(D^{tr})$.

So that the meta-training process is trying to optimize the prior parameters such that adaptation leads to good performance.

To evaluate a meta-learning algorithm, the dataset used for meta-learning is proposed to be used as few-shot discriminative and few-shot generative problems. Initial few-shot learning approaches with Bayesian models, non-parametrics. Other datasets used for few-shot image recognition: Minilmagenet, CIFAR, CUB, CelebA, etc.

Few-shot learning problem: 5-way, 1-shot image classification means 5-classes, 1-example per class. Which means that given one example of five classes to classify new examples. Few-shot learning could be seen as a subset of meta-learning.

The Mechanistic View - Supervised learning v.s. Meta-Supervised learning

	Supervised learning	Meta-Supervised learning
Inputs	x	$f(x;\theta)$
Outputs	$D^{tr} = \{(x,y)_{1:k}\}$	$f(D^{tr}, x_{test}; \theta)$
Data	$D = \{(x,y)_i\}$	$D_{meta-train} = \{D_i\}, D_i : \{(x,y)_j\}$

k means k input output pairs for a k-shot learning problem. In order to make predictions about new test data point X_{test}

Meta-learning reduces the problem to the design and optimization of f. The assumption is that the distribution of training and testing set is the same.

General recipe

How to design a meta-learning algorithm? First, choose a form of $p(\phi_i|D_i^{tr}, \theta)$. Second, choose how to optimize θ with respect to max-likelihood objective using $D_{meta-train}$. Can we treat $p(\phi_i|D_i^{tr}, \theta)$ as an inference problem? Neural networks are good at inference.

Black-Box Adaptation Key idea: Train a neural network to represent $p(\phi_i|D_i^{tr}, \theta$. For now, Use deterministic (point estimate) $\phi_i = f_\theta(D_i^{tr})$

Train with standard supervised learning: $\max_\theta \sum_{T_i} \sum_{(x,y)\sim D_i^{test}} log\, g_{\phi i}(y|x)$, where $L(\phi_i, D_i^{test}) = \sum_{(x,y)\sim D_i^{test}} log\, g_{\phi i}(y|x)$.

So that $\max_\theta \sum_{T_i} \sum_{(x,y)\sim D_i^{test}} log\, g_{\phi i}(y|x) = \max_\theta \sum_{T_i} L(\phi_i, D_i^{test})$.

- Sample task T_i (or mini batch of tasks)
- Sample disjoint datasets $D_i^{tr}, D_i^{test}$from D_i
- Compute $\phi_i \leftarrow f_\theta(D_i^{tr})$
- Update θ using $\nabla_\theta \mathscr{L}(\phi_i, D_i^{test})$

A Example

The complete Meta-Learning Optimization

Some Meta-Learning terminology

Closely related problem settings

General recipe of meta-learning algorithms

Black-box adaptation approaches

Motivation: If we want to infer all of the parameters of a neural network, having a neural network to output them is not a scalable way to do that.

Bayesian Meta-learning

5.3.3 GPT-3

Human naturally do not require large supervised datasets to learn most language tasks. A brief directive in natural language or at most a tiny number of demonstrations is sufficient to enable a human to perform a new task to a reasonable degree of competence. One potential way to address these issues is Meta-Learning. While training, the model develops a broad set of skills and pattern recognition abilities. While inferring, the model uses those abilities to recognize or adapt desired task.

Previously, language pre-training model[63] attempts to fulfill this by using the text input as a/a few task demonstration(s) to condition the model and then predicting what comes next to complete the further instances of the task. Another potential way to address these issues is increasing the capacity of transformer language models [62], [15], [63], [69], [64], and [52]. This has improved in text synthesis and/or performances of NLP tasks. In the future it is plausible that the language pre-training model [63] with scale [33] could have similarly strong performance as Meta-Learning, since the language pre-training model can absorb many skills and tasks within its parameters.

Task-agnostic cannot be solved through current benchmark solution by fine-tuning a pre-training model on a large corpus of text because of the limited amount of texts samples. However, training GPT-3 [7] - an autoregressive language model with 175 billion parameters without any gradient updates or fine-tuning - can achieve state-of-the-art performance of prior fine-tuning approaches by testing its performance in the few-shot setting, which is tasks and few-shot demonstrations specified purely via text interaction with the model.

The model architecture of GPT-3 is similar with GPT-2 [63], including the modified initialization, pre-normalization, and reversible tokenization. The only exception is that using alternating dense and locally banded sparse attention patterns in the layers of the transformer, similar to the Sparse Transformer [11].

GPT-3 achieves strong performance on many NLP datasets, such as, translation, question-answering, cloze tasks, and several tasks that require rapid reasoning or domain adaptation including unscrambling words, performing arithmetic, and using novel words in a sentence after seeing them defined only once. Specifically, for each task, GPT-3 is evaluated under three conditions: (a) few-shot learning allowing 10 to 100 demonstrations fit into the model's context window; (b) one-shot learning allowing only one demonstration; (c) zero-shot learning allowing no demonstrations and only one instruction in natural language is given to the model.

Compared with zero-shot and one-shot, few-shot performance is more apparently well as model capacity goes more larger. That is to say larger models are more proficient meta-learners.

In addition, GPT-3 can generate synthetic new articles which is hard to distinguish from human-generated articles. Data contamination - a problem of potentially training test datasets samples while training high capacity models on datasets such as Common Crawl, since such test samples often exist on the web - on most datasets has a minimal effect on GPT-3's performance.

However, few-shot learning, even at the scale of GPT-3, facing methodological issues, so that it cannot perform well on datasets of some tasks including natural language inference ANLI dataset, reading comprehension datasets RACE and QuAC. Those are further directions of study which needs highly attention.

5.4 Mixture Models

A Mixture Models is a collection of distributions $D_1, ..., D_k$ and weights $w_1, ...w_k$. To sample from a mixture model, we draw i with probability w_i, where $_i \in \{1, ..., k\}$, and then draw a random sample from D_i, where $_i \in \{1, ..., k\}$. Thus, each cluster in the data corresponds to a different distribution D_i in the mixture, and the weights w_i correspond to the fraction of points from each cluster in the entire dataset.

In the problem of learning mixture models, we are given samples from a mixture model, and our goal is to (a) learning the parameters of the distributions D_i and (b) classify each sample, according to its source distribution.

The distribution in the mixture are very close together in space, then the mixture will be hard to learn. A solution **separation condition**[14] is first introduced, which promises to ensure that every pair of distributions in the mixture are far apart. Given a mixture with a certain separation, the goal of the algorithm designer is to learn the correct clustering of the data.

Separation is defined as the minimum distance between the means of any pairs of distributions in the mixture, as a function of a standard deviation.

Examples of mixture models include Random Projections[14]; EM[13]; Distance thresholding[67]; PCA Projections[74][32][1]; Canonical Correlations[9]; and Isotropic PCA[8]. (This section will be further expended.)

Learning Mixtures with No Separation Condition, i.e. in the absence of a lower bound on the separation, the problem of learning mixture models is considered to be quite hard, when there are more than two clusters. In this case, one is again samples from a mixture model, and the goal is to produce a mixture which has KL-Divergence at most ϵ to the true mixture.

5.5 Attention Mechanism

Transformer[73] is a model architecture based entirely on an attention mechanism.

Transformer is a new simple network architecture restricted to NLP, based solely on Attention layers, which are CNNs that capture the relevance of any sequence element to each other, dispensing with recurrence and convolutions entirely, which previously dominant sequence transduction models. This achieved a better state-of-the-art results and markedly faster than previous RNN models.

The combination and application of Attention mechanism is very powerful. The combination of techniques in MobileNets[29], which achieves more efficient models, and transformers, which achieves more efficient training, could achieve efficient training and inference. GPT-2, GPT-3, and BERT[15] are novel architectures in NLP which are descendants of Transformer. Attention is not only useful in NLP, but also useful in generative models e.g., Self-Attention GAN[91].

MobileNets is a set of low-parameter networks, so that is is efficient for real-time applications. The core idea of MobileNet and is to decompose expensive operations into a set of smaller (and faster) operations. MobileNetV2[66] and MobileNetV3[28] are descendent of MobileNets making great progress on improving accuracy and reduciing size; SqueezeNet[31] and a set of ConvNets[38] are variant of MobileNets making great progress on reducing size.

Transformer could be further improved in some ways. For example, Transformer is inefficient while training, which is further improved by Reformer[35].

An alternative way of improving efficient, which is not a variant or descendent of Transformer is Single Headed Attention RNN[50], which is a variation of LSTMs. It achieves state-of-the-art results on enwik8 without intensive hyper parameter optimization, so that it is efficient. Additionally, this Attention mechanism is also readily extended to large contexts with minimal computation.

5.5.1 Transformer

Transformer mechanism uses stacked self-attention and point-wise fully connected layers for both encoder and decoder. Encoder maps an input $(x_1, ..., x_n)$, a sequence of symbol representations, to a sequence of continuous representations, $z = (z_1, ..., z_n)$. Decoder uses z generate an output sequence of symbols, $(y_1, ..., y_n)$. The sequence of output are generated one element at a time, and each symbol is generated based on the previous symbol (auto-regressive).

Encoder is composed of a stack of 6 identical layers. Each layer has two sub-layers, the first is a multi-head self-attention mechanism and the second is a position-wise fully connected feed-forward network. Transformer emploies a residual connection[26] around each of the two sub-layers and followed by layer normalization[3]. Thus, the output of each sub-layer is $LayerNorm(x + Sublayer(x))$, where $Sublayer(x)$ is the function implemented by the sub-layer itself. All sub-layers and the embedding layers produce outputs of dimension $d_{model} = 512$ in order to facilitate residual connections.

Decoder is also composed of a stack of six identical layers. The only differences between Decoder and Encoder are: (1)a third sub-layer is inserted into the two sub-layers. It performs multi-head attention over the output of the encoder stack. Residual connections around each of the sub-layers followed by layer normalization are also employed in this third sub-layer like the others two sub-layers. (2) The self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions. This masking ensures the predictions for position i, where $i \geqslant 2$, depending only on the known output at position $i - 1$.

Attention function is mapping a query and a set of key-value pairs to an output. The query, keys, values, and output are all vectors. The output is a weighted sum of values, which is

computed by a compatibility function of the query with the corresponding key.

Scaled Dot-Product Attention:

Input consists of

- queries with dimension d_k,
- keys with dimension d_k, and
- values with dimension d_v.

Output of the matrix of attention function is

$$DotproductAttention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

, where matrices Q, K, V are sets of queries, keys, and values packed together.

Multi-Head Attention:

The input is the same as input of Scaled Dot-Product Attention.

The matrix of outputs is

$$MultiHeadAttention(Q, K, V) = Concat(head_1, ..., head_h)W^O$$

$$head_i = DotproductAttention(QW_i^Q, KW_i^K, VW_i^V)$$

, where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_q}, W_i^K \in \mathbb{R}^{d_{model} \times d_k}, W_i^V \in \mathbb{R}^{d_{model} \times d_v}$

Self-attention is an attention mechanism using different positions of a single sequence to compute a representation in that sequence.

5.5.2 Self-Attention GAN

GAN-based models are computationally inefficient for modeling long-range dependencies in images. Because those models is built using convolutional layers, which processes information in a local neighborhood. Self-Attention GAN (SAGAN) efficiently addressed this inefficiency by introducing self-attention to the GAN framework to make the model non-local based and adapt a widely separated spatial regions.

Model Architecture

- First, The image features from the previous hidden layer $x \in \mathbb{R}^{C \times N}$ are transformed into two feature spaces f, g, where C is the number of channels and N is the number of feature locations of features from the previous hidden layer.
- Second, calculate the attention, where $f(x) = W_f x, g(x) = W_g x$
- Third, calculate $\beta_{j,i}$, which indicates the extent that the model attends to the i_{th}, where $s_{ij} = f(x_i)^T g(x_j)$,

$$\beta_{j,i} = \frac{exp(s_{ij})}{\sum_{i=1}^{N} exp(s_{ij})}$$

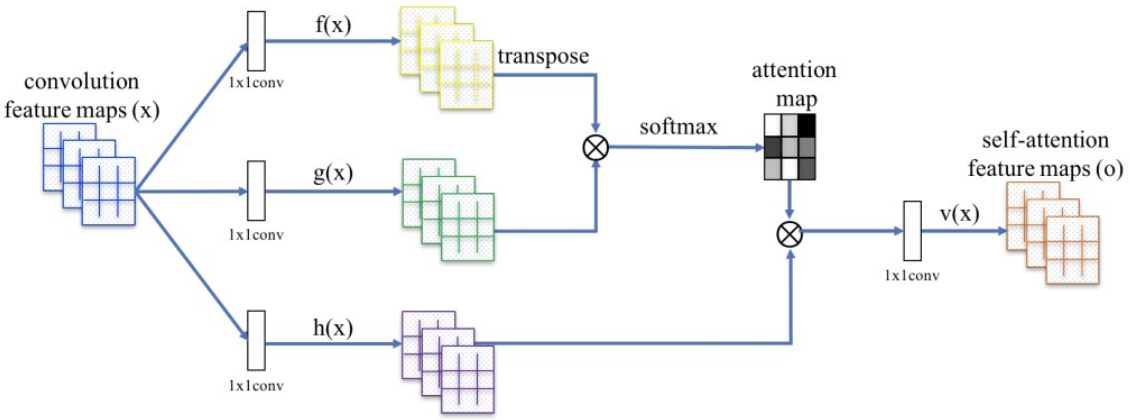


Figure 2: **SAGAN Algorithm**. The $\otimes$ denotes matrix multiplication.[91]

- The output of the attention layer is $o = (o_1, o_2, ..., o_j, ..., o_N) \in \mathbb{R}^{C \times N}$, where,

$$o_j = v\Big(\sum_{i=1}^{N} \beta_{j,i} h(x_i)\Big), h(x_i) = W_h x_i, v(x_i) = W_v x_i.$$

  $W_g \in \mathbb{R}^{\overline{C} \times C}, W_f \in \mathbb{R}^{\overline{C} \times C}, W_h \in \mathbb{R}^{\overline{C} \times C}, W_v \in \mathbb{R}^{\overline{C} \times C}$ are the learned weight matrices, which are implemented as 1x1 convolutions.

- The final output is given by $y_i = \gamma o_i + x_i$, where γ is a learnable scalar, which is initialized as 0. γ initialized as 0 allows the network to first rely on the cues in the local neighborhood, then gradually learns to assign more weight to the non-local evidence. The reason of doing this is to first learn easy task and then progressively increase the complexity of the task.

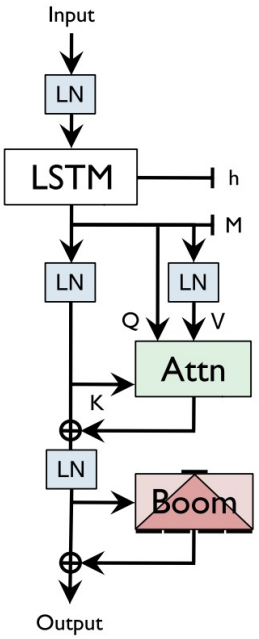
5.5.3 Single Headed Attention RNN:

This is an upgrade of the AWD-LSTM[51]. The model contains: a trainable embedding layer, at least one layer of a stacked single head attention recurrent neural network (SHA-RNN), and a softmax classifier. Both embedding and softmax classifier utilize tied weights. The model uses a single head of attention, and a Boom layer which means a modified feedforward layer similar to that in a Transformer.

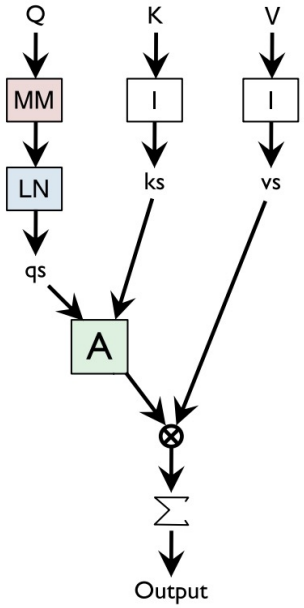
The attention mechanism has been simplified in SHA-RNN, (since no-one proves that tens of attention is necessary and beneficial) which saves lots of memory space and avoids a great deal of computation. This helps extend memory window indefinitely with minimal overhead.

The Boom layer is related to the large feed forward layer in Transformers. To minimize computation and replace an entire matrix of parameters into traditional down-projection layers, the Boom layer is rearranged in the following way:

- First, like what other Transformers did in the Boom layer, where a vector, $v \in \mathbb{R}^H$, is taken to do matrix multiplication with GeLU activation function to produce a vector, $v \in \mathbb{R}^{N \times H}$.
- Second, breaking u into N vectors, $u_1, ..., u_n \in \mathbb{R}^H$.
- Third, summing those together to produce $w \in \mathbb{R}^H$.



(a) The SHA-RNN is composed of a pointer based attention and a "Boom" feed-forward with a sprinkling of layer normalization. The persistent state is the RNN's hidden state h as well as the memory M concatenated from previous memories.



(b) The attention mechanism within the SHA-RNN is highly computationally efficient. The only matrix multiplication acts on the query. The A block represents scaled dot product attention, a vector-vector operation. The operators $\{qs, ks, vs\}$ are vector-vector multiplications and thus have minimal overhead. We use a sigmoid to produce $\{qs, ks\}$, where $vs = \theta(W^f v) \dot{} tanh(W^c v)$.

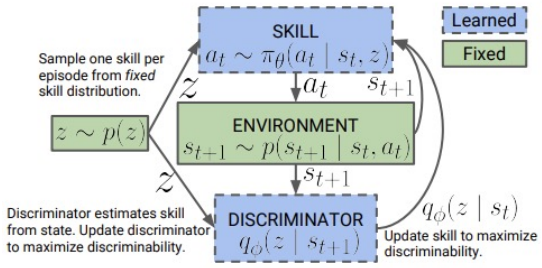
Figure 3: SHA-RNN Architecture

5.6 Diversity Mechanism

DIVERSITY IS ALL YOU NEED [18] (DIAYN) proposes a new method for learning useful skill using reinforcement learning without a reword function. On a variety of simulated robotic tasks, this method results in the unsupervised emergence of diverse skills e.g., walking and jumping. This method is able to solve the benchmark task despite never receiving the true task reward. Also the learned skills could solve the task in a distinct manner. This work also shows that unsupervised discovery of skills can serve as an effective pertraining mechanism for overcoming challenges of exploration and data efficiency in reinforcement learning.

5.6.1 Learning Skills without a Reward Function

The procedure of DIAYN is that: (1) first, train multiple skills and distinguish them by states; (2) second, encourage each skill to have a high entropy in order to keep discriminable. However, two skills that are largely distinguishable in states does not necessary obviously diverse in skills. So we should train skills as random as we can instead of select skills that are largely distinguishable in state.



Algorithm 1: DIAYN

while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
  for $t \leftarrow 1$ **to** *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.

Figure 4: **DIAYN Algorithm**: We update the discriminator to better predict thee skill, and update the skill to visit diverse states that make it more discriminable.

First, we sample a skill from uniform distribution. The skill is corresponding to a policy. Then we sample actions based on the policy, so that we have a state of that action and skill. Then we train the discriminator, which is only given access to the current state.

S is a random variable for a state and A is a random variable for an action. Z is a random variable for a skill. $Z \sim p(z)$ is a latent variable on which we condition policy. $I(.;.)$ refers to mutual information. $I(S;Z)$ is the mutual information between S and Z. $H[.]$ refers to Shannon entropy. To ensure the skill control which states the agent visits, we maximize mutual information between skills and states, $I(S;Z)$. To ensure that it is state that distinguishes skills, instead of action, we minimize the mutual information between skills and actions given the state, $I(A;Z|S)$. A mixture of policies is a combination of skills and $p(z)$, we maximize the entropy $H[A|S]$ of this mixture policy.

$$
\begin{aligned}
F(\theta) &\triangleq I(S;Z) + H[A|S] - I(A;Z|S) \\
&= (H[Z] - H[Z|S]) + H[A|S] - (H[A|S] - H[A|S,Z]) \\
&= H[Z] - H[Z|S] + H[A|S,Z]
\end{aligned}
$$

$H[Z]$ encourages prior distribution over $p(z)$ to have high entropy. We fix $p(z)$ to be uniform, guaranteeing that is has maximum entropy. $H[Z|S]$ suggests that it should be easy to infer the skill z from the current state. $H[A|S,Z]$ suggests that each skill should act as randomly as possible, which we achieve by using a maximum entropy policy to represent each skill. As we cannot integrate over all states and skills to compute $p(z|s)$ exactly, we approximate this posterior with, $q\phi(z|s)$, a learned discriminator. By Jensen's Inequality, replacing $p(z|s)$ with $q\phi(z|s)$ gives us a variational lower bound $\varrho(\theta,\phi)$ on our objective $F(\theta)$.

$$
\begin{aligned}
F(\theta) &= H[A|S,Z] - H[Z|S] - H[Z] \\
&= H[A|S,Z] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[logp(z|s)] - \mathbb{E}_{z\sim p(z)}[logp(z)] \\
&\geqslant H[A|S,Z] + \mathbb{E}_{z\sim p(z),s\sim\pi(z)}[logp(z|s) - logp(z)] \\
&\triangleq \varrho(\theta,\phi)
\end{aligned}
$$

6 Using Meta-Learning to Address the Task-Agnostic Problem in Natural Language Understanding

6.1 Introduction

Human beings can generalize from a single example of a task to similar tasks. However, computers need to effectively see thousands of examples to learn to generalize to similar tasks. To learn computers need to be given labeled datasets, which include both correct and incorrect examples, properly labeled. The limited amount of available labeled datasets makes it difficult for machine learning to achieve human-level performance. Meta-Learning, which is known as "learning to learn", can address this difficulty. This means that meta-Learning enables learning with a few training samples and enables adaptation across domains or in a single constantly changing domain. Meta-Learning has been explored and used in robotics and computer vision, but it has not been sufficiently explored in natural language processing. This research aims to explore how to use meta-Learning in addressing task-agnostic problems in Natural Language Understanding.

6.2 Symbolic Approach

6.2.1 Semantic Parsing Framework

The task is defined as given a reference table t and a question x, we predict an answer y. For prediction, we first convert the reference table t to a knowledge graph of words w. Second, we generate a set of candidate logical forms $\mathcal{Z}_x$ by parsing x using w. For each $z \in \mathcal{Z}_x$, we can

get a corresponding answer denotation $[z]_w$ by executing z in w. We extract a feature vector $\phi(x, w, z)$ for each $z \in \mathcal{Z}_x$ and define a log-linear distribution over the candidates:

$$p_\theta(z|x, w) \propto \exp\{(\theta^\top \phi(x, w, z)\},$$

where θ is the parameter vector. We finally choose z with the highest p_θ and execute z on w to predict the answer denotation $y = [z]_w$.

For training, given the training examples $D = \{(x_i, t_i, y_i)\}_{i=1}^N$, we seek θ that maximizes the regularized log-likelihood of the correct y_i marginalized over logical forms z.

Formally, we maximize the objective function:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \log p_\theta(y_i|x_i, w_i) - \lambda||\theta||_1,$$

where w_i is deterministically generated from t_i, and

$$p_\theta(y|x, w) = \sum_{z \in Z_x; y=[z]_w} p_\theta(z|x, w).$$

6.2.2 Knowledge Representation

We deterministically convert the table t into a knowledge graph k, Figure 5. Table rows become row nodes, strings in table cells become entity nodes, and table columns become directed edges from the row nodes to the entity nodes of that column. The column headers are used as edge labels for these row-entity relations.The knowledge graph representation is convenient for three reasons.

First, we can encode different forms of entity normalization in the graph. The edges correspond to different normalization methods from the entity nodes. For example, the node 1900 will have an edge called Date to another node $1900/DD/MM$ of type date. Moreover, these normalization nodes also aid learning by providing signals on the appropriate answer method. For instance, we can define a feature that associates the phrase "how many" with a logical form that says "traverse a row-entity edge, then a number edge" instead of just "traverse a row-entity edge."

Second, the graph representation could handle various logical phenomena via graph augmentation. There are two special edges on each row node. The Next edge pointing to the next row node, the questions can be answered by traversing the Next edge. And an Index edge pointing to the row index number $(0, 1, 2, ...)$, which is used to answer questions, such as, for instance, "What is the next?" or "Who came before?".

Third, with a graph representation, we can query it directly using a logical formalism for knowledge graphs.

6.2.3 Logical Forms

We use lambda dependency-based compositional semantics [42], or lambda DCS, to construct our logical forms. Each lambda DCS logical form is either a unary or a binary. The basic unary elements are singletons, which represent values, for instance, "China" and "30". The basic binaries are relations, which represent relationships, for instance, "City" maps rows to city entities, "Next" maps rows to rows, and ">=" maps numbers to numbers. Logical forms can be combined into larger ones via various operations based on unary and binary rules.

Year	City	Country	Nations
1896	Athens	Greece	14
1900	Paris	France	24
1904	St. Louis	USA	12
...	...	...	...
2004	Athens	Greece	201
2008	Beijing	China	204
2012	London	UK	204

x_1: *"Greece held its last Summer Olympics in which year?"*
y_1: {2004}

x_2: *"In which city's the first time with at least 20 nations?"*
y_2: {Paris}

x_3: *"Which years have the most participating countries?"*
y_3: {2008, 2012}

x_4: *"How many events were in Athens, Greece?"*
y_4: {2}

x_5: *"How many more participants were there in 1900 than in the first year?"*
y_5: {10}

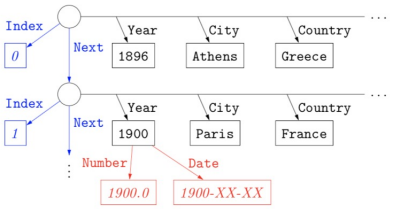


Figure 5: An example of the knowledge graph [59]. Circular nodes are row nodes. Number and Date are different entity normalization nodes.

6.2.4 Parsing Algorithm

Given the knowledge graph w, we describe how to parse the utterance x into a set of candidate logical forms Z_x.

Because of the mismatch problem in the standard Chart parser, which anchoring each predicate in the logical form to tokens in the utterance via lexical rules. We use the Floating parser, which makes parsing more freely. Floating parser [59] allows logical form predicates to be generated independently from the utterance.

The *floating cells* represented as (c, s), where c is category and s is logical form size. It follows three rules:

- $(TokenSpan, i, j)[s] \rightarrow (c, 1)[f(s)]$, this allows us to keep track of the category c and its size 1.
- $\emptyset \rightarrow (c, 1)[f()]$, this allows us to construct logical form representing relation, using rules independent of the utterance. For instance, using rule $\emptyset \rightarrow (c, 1)[f()]$ to represent a table relation 'Country' in cell $(Relation, 1)$.
- $(c_1, s_1)[z_1] + (c_2, s_2)[z_2] \rightarrow (c, s_1 + s_2 + 1)[f(z_1, z_2)]$, this allows us to perform composition, where the induction is on the size s of the logical form.

The floating parser is very flexible: it can skip tokens and combine logical forms in any order.

6.3 Neural Network Approach

To learn multiple tasks at once to solve the task-agnostic problem, the model that we choose to use is Meta-Learning. This is because of the three properties of the Meta-Learning, which made Meta-Learning more robust than any other deep learning models while addressing task-agnostic problems or unforeseen tasks.

Recently meta-learning has obtained high attention and great success in many domains. Most common uses of the technique are for hyper-parameter [47] and neural network optimization [39, 10], finding good network architectures [93, 4, 56], few-shot image recognition [20, 65, 25], and fast reinforcement learning [17, 83, 20].

There are three common types of meta-learning approaches: metric-based, model-based, and optimization-based.

Metric-based meta-learning is designed to learn a metric function, the idea is similar to nearest neighbor algorithms, for instance, k-nearest neighbors algorithm, k-means clustering, and kernel density estimation. The goal is to learn a good kernel for a specific task [36, 75, 71, 70]. Model-based meta-learning is designed specifically for fast learning, which means updating its parameters rapidly with a few training steps. One way of achieving this is to be controlled by another meta-learner model [68, 54]. Another way is is make a specific internal architecture to achieve its goal [85, 23]. It makes no assumptions on $P_\theta(y|\mathbf{x})$.

Optimization-based meta-learning is designed to achieve good performance by learning with few examples. Because of the reality that there are lots of

According to those three approaches, their corresponding properties, and the property of our task, we choose the optimization-based approach to address our task.

6.3.1 Meta-Learning

A usual meta-learning model is trained over a variety of learning tasks and optimized for the best performance on a distribution of tasks, including unseen tasks. Each task is associated with a dataset D, containing both feature vectors and target labels. The optimal model parameters are: $\theta^* = \arg\min_\theta \mathbb{E}_{D\sim p(D)}[\mathscr{L}_\theta(D)]$, where each data sample is a set of data. K-shot N-class classification is a supervised instance of meta-Learning. This means that the support set contains K samples in total for each of N classes. The dataset D is often split into two parts, a learning and prediction set of S, and a training and testing set of B.

A dataset $D = \{(x_i, y_i)\}$, where (x_i, y_i) is a pair of feature vectors and labels. Each label $y_i \in L^{label}$, where L^{label} is a known label set.

Assume that classifier f_θ with optimal parameter θ outputs a probability $P_\theta(y|\mathbf{x})$ of a data point, given the feature vector $\mathbf{x}$ belonging to the class y.

The optimal parameters θ should the probability of true labels across training batches $B \subset D$:

$$\begin{aligned}\theta^* &= \arg\max_\theta \mathbb{E}_{(\mathbf{x},y)\in D}[P_\theta(y|\mathbf{x})] \\ &= \arg\max_\theta \mathbb{E}_{(B\subset D}[\sum_{(\mathbf{x},\mathbf{y})\in\mathbf{B}} P_\theta(y|\mathbf{x})]\end{aligned}$$

The few-shot classification has two goals:

- reduce the prediction error on task-agnostic dataset, i.e., a set of data samples with unknown labels.
- learn fast a subset of training and prediction sets.

The modified model with the optimization procedure in order to achieve the fast learning could be represented as follows.

We select a subset from the set of target labels, $L \subset L^{label}$. Then we have the learning and prediction set $S^L D$ and training and testing set $B^L D$, where $y \in L, \forall(x, y) \in S^L, B^L$. The support set is part of the model input. Then we consider each pair of sampled dataset (S^L, B^L) as one data point. The trained model can generalize to other datasets. The representation of meta-learning is

$$\theta = \arg \max_{\theta} \mathbb{E}_{L \subset \mathscr{L}}[\mathbb{E}_{S^L \subset D, B^L \subset D}[\sum_{(x,y) \in B^L} P_\theta(x, y, S^L)]].$$

The idea is similar to use a pre-trained language model on big text corpora with only a limited set of task-specific data samples available. Meta-Learning takes this idea one step further, rather than fine-tuning on a single down-steam task. It optimizes the model to be good at all or at least many.

Another point of view of meta-learning could be a two stages update.

- A classifier f_θ is the "learner" model, trained for operating a given task;
- A optimizer g_ϕ learns to update the learner model's parameters θ via the support set S, $\theta \to \theta' = g_\phi(\theta, S)$.
- In the final optimization step, maximize the parameter $\theta =$

  $$\mathbb{E}_{L \subset \mathscr{L}}[\mathbb{E}_{S^L \subset D, B^L \subset D}[\sum_{(\mathbf{x},y) \in B^L} P_{g_\phi(\theta, S^L)}(y|\mathbf{x})]]$$

  by updating both θ and ϕ.

6.3.2 Optimization-based meta-Learning

Now we introduce some Optimization-based Meta-Learning (OBML) architectures based on meta-learning, which is a good fit for our task.

LSTM-based Meta-Learner [**DBLP:conf/iclr/RaviL17**], is to learn the optimization algorithm used to train another learner neural network classifier in the few-shot regime [1], which means to efficiently update the learner's parameters using a few support set, in order to achieve the goal of adapting to the new task quickly.

Let's denote the learning model for operating the task as M_Θ with parameters Θ, the meta-learner as R_ϕ with parameters ϕ, and the loss function $\mathcal{L}$. The update for the learner's parameters ϕ_t at time step t, with a learning rate r_t is: $\phi_t = \phi_{t-1} - r_t \nabla_{\phi-1} \mathcal{L}_t$. The training process mimics the testing process. For each training epoch, we first sample a dataset $D = (D_{train}, D_{test}) \in \hat{D}_{meta-train}$, and then sample mini-batches out of D_{train} to update θ for T rounds. The final state of the learner parameter θ_T is for training the meta-learner on the test data D_{test}.

Model-Agnostic Meta-Learning (MAML) [20] is a general optimization algorithm. Let's denote learning model is f_θ with parameters θ. Given a task τ_i and its associated dataset $(D^{(i)}_{train}, D^{(i)}_{test})$, we can update the model parameters by one gradient descent step, $\theta'_i = \theta -$

[1] A K-shot N-class classification task means the support set contains K labelled examples for each of N classes.

Algorithm 3 LSTM-based Meta-Learner

Input: Meta-training set $\mathscr{D}_{-}$, Learner M with parameters θ, Meta-Learner R with ϕ.

$\phi_0 \leftarrow$random initialization $d = 1, n$ $\mathscr{D}$, $\mathscr{D} \leftarrow$ random dataset from $\mathcal{D}_{-}$ $\theta_0 \leftarrow c_0$ $t = 1, T$ $\mathbf{X}_t, \mathbf{Y}_t \leftarrow$ random batch from $\mathscr{D}$ $\mathcal{L}_t \leftarrow \mathcal{L}(M(\mathbf{X}_t; \theta_{t-1}), \mathbf{Y}_t)$ $c_t \leftarrow R((\nabla_{\theta_{t-1}} \mathcal{L}_t; \mathcal{L}_t), \phi_{d-1})$ $\theta_t \leftarrow c_t$ $\mathbf{X}$, $\mathbf{Y} \leftarrow \mathscr{D}$ $\mathscr{D} \leftarrow \mathscr{L}(M(\mathbf{X}; \theta_T), \mathbf{Y})$ Update ϕ_d using $\nabla_{\phi_{d-1}} \mathscr{L}$

Algorithm 4 Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: α, β : step size hyper-parameters

random initialize θ not done Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$ $\mathcal{T}_i$ Evaluate $\nabla_\theta \mathscr{L}_{\mathcal{T}}(f_\theta)$ with respect to K examples Compute adapted parameters with gradient descent: $\theta_i' = \theta - \alpha \nabla_\theta \mathscr{L}_{\mathcal{T}}(f_\theta)$ Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'})$

$r\nabla_\theta \mathscr{L}^{(0)}_{\tau_i}(f_\theta)$, where $\mathscr{L}^{()}$ is the loss computed using the mini data batch with id (0), or more gradient descent steps,

$$\begin{aligned}\theta^* &= \min_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}^{(1)}(f_{\theta *_i}) \\ &= \min_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}^{(1)}(f_{\theta - r\nabla_\theta \mathcal{L}^{(0)}_{\tau_i}(f_\theta)})\end{aligned}$$

In the case of Figure 6, the optimal θ^* is more likely be θ_2^* choosing from θ_1^*, θ_2^*, and θ_3^*. For each task τ_i, we finally update the θ for it,

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_{\tau_i \sim p(\tau)} \mathcal{L}^{(1)}(f_{\theta *_i}),$$

where the loss function $\mathcal{L}^{(i)}$ means the loss function of a sample of data batch with id(i).

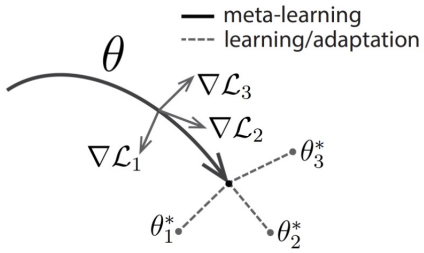


Figure 6: Diagram of MAML, which optimizes for a representation θ that can quickly adapt to new tasks [21]. θ_i^* is the learning parameter learned from mini data batch with id(i).

First-Order MAML (FOMAML) simplifies the MAML by omitting second derivatives in MAML, to reduce the computational complexity of MAML. Which means simplifying Step8 in Algorithm4, $\theta_{meta} \leftarrow \theta_{meta} - \beta g_{\text{MAML}}$, where

$$g_{\text{MAML}} = \nabla_{\theta_k} \mathcal{L}^{(j)}(\theta_k) \prod_{i=1}^{k} (\mathcal{I} - r\nabla_{\theta_{i-1}}(\nabla_\theta \mathcal{L}^{(j-1)}(\theta_{i-1}))),$$

to $\theta_{meta} \leftarrow \theta_{meta} - \beta g_{\text{FOMAML}}$, where

$$g_{\text{FOMAML}} = \nabla_{\theta_k} \mathcal{L}^{(j)}(\theta_k).$$

Algorithm 5 Reptile

Initialize θ, the vector of initial parameters iteration = 1,2,... Sample tasks $\tau_1, \tau_2, ..., \tau_n$ $i = 1, 2, ..., n$ Compute $W_i = \text{SGD}(\mathcal{L}_{T_i}, \theta, k)$ Update $\theta \leftarrow \theta - \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$

(j) represents the id of a sample of data batch.

Reptile [57] is a simple meta-learning optimization algorithm. Both MAML and Reptile optimize via gradient descent and both are model-agnostic.

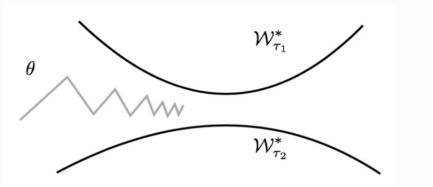


Figure 7: Diagram of the Reptile algorithm. The sequence of iterations moves alternately towards two optimal solution manifolds $\mathcal{W}^*_{\tau_1}$ and $\mathcal{W}^*_{\tau_2}$, and converges the learning parameter θ to the point that minimize the average squared distance.

Assume a task $\tau \sim p(\tau)$ has a manifold of optimal network configuration $\mathcal{W}^*_\tau$. The model f_θ achieves the best performance for task τ when θ lays on the surface of $\mathcal{W}^*_\tau$. To find a solution that is good across tasks, we would like to find a parameter θ close to all the optimal manifolds of all tasks

$$\theta = \min_\theta \mathbb{E}_{\tau \sim p(\tau)} [\frac{1}{2} \text{dist}(\theta, \mathcal{W}^*_\tau)^2],$$

where

$$\text{dist}(\theta, \mathcal{W}^*_\tau) = \text{dist}(\theta, \mathcal{W}^*_\tau(\theta)),$$

where

$$\mathcal{W}^*_\tau(\theta) = \min_{W \in \mathcal{W}^*_\tau} \text{dist}(\theta, W).$$

Thus, the stochastic gradient update step is

$$\begin{aligned} \theta &= \theta - r \nabla_\theta [\frac{1}{2} \text{dist}(\theta, \mathcal{W}^*_\tau)^2] \\ &= \theta - r(\theta - \mathcal{W}^*_{\tau_i}(\theta)) \\ &= (1 - r)\theta + r \mathcal{W}^*_{\tau_i}(\theta). \end{aligned}$$

We cannot compute exactly the closest point on the optimal task manifold $\mathcal{W}^*_{\tau_i}(\theta)$, but Reptile approximates it using $\text{SGD}(\mathcal{L}_{T_i}, \theta, k)$.

6.3.3 MAML vs FOMAML vs Reptile

To represent the connection and differences among MAML, FOMAML, Reptile, we compare their update rule in the case of two gradient steps $k = 2$. We assume that $g_j^{(i)} = \nabla_\theta \mathcal{L}^{(i)}(\theta_j)$ and $h_j^{(i)} = \nabla^2_\theta \mathcal{L}^{(i)}(\theta_j)$ for simplification of the two gradient steps, and also assume that there are only two different mini-batches of data samples with losses $\mathcal{L}^{(0)}$ and $\mathcal{L}^{(1)}$. Also by Taylor

expansion, we get the following three formulas of gradient update:

$$
\begin{aligned}
g_{\text{MAML}} &= g_0^{(1)} - r h_0^{(1)} g_0^{(0)} - r h_0^{(0)} g_0^{(1)} + O(r^2) \\
g_{\text{FOMAML}} &= g_0^{(1)} - r h_0^{(1)} g_0^{(0)} + O(r^2) \\
g_{\text{Reptile}} &= g_0^{(0)} + g_0^{(1)} - r h_0^{(1)} g_0^{(0)} + O(r^2)
\end{aligned}
$$

During training, we often average over multiple data batches. The expectation $\mathbb{E}_{\tau,0,1}$ is averaged over data batches, ids(0) and (1) in our case, for task τ. Thus we get

- The average gradient of task loss,

  $$\mathbb{E}_{\tau,0,1}[g_0] = \mathbb{E}_{\tau,0,1}[g_0^{(0)}] = \mathbb{E}_{\tau,0,1}[g_0^{(1)}].$$

  We expect to improve the model parameter to achieve better *task performance* by following this direction.

- The gradient that increases the inner product of gradients of two different mini batches for the same task,

  $$\mathbb{E}_{\tau,0,1}[h_0 g_0] = \frac{1}{2}\mathbb{E}_{\tau,0,1}[h_0^{(1)} + g_0^{(0)}] = \frac{1}{2}\mathbb{E}_{\tau,0,1}[\nabla_\theta g_0^{(0)} g_0^{(1)}].$$

  We expect to improve the model parameter to achieve better *generalization* over different data by following this direction.

Therefore, we get

$$
\begin{aligned}
\mathbb{E}_{\tau,0,1}[g_{\text{MAML}}] &= \mathbb{E}_{\tau,0,1}[g_0] - 2r\mathbb{E}_{\tau,0,1}[h_0 g_0] + O(r^2) \\
\mathbb{E}_{\tau,0,1}[g_{\text{FOMAML}}] &= \mathbb{E}_{\tau,0,1}[g_0] - r\mathbb{E}_{\tau,0,1}[h_0 g_0] + O(r^2) \\
\mathbb{E}_{\tau,0,1}[g_{\text{Reptile}}] &= 2\mathbb{E}_{\tau,0,1}[g_0] - r\mathbb{E}_{\tau,0,1}[h_0 g_0] + O(r^2)
\end{aligned}
$$

In conclusion, FOMAML is able to obtain a similar performance as the full version of MAML. Both MAML and Reptile aim to optimize for the same goal, better task performance, $E[g_0]$, and better generalization, $E[h_0 g_0]$.

6.3.4 Symbolic-Neural Approach

Semantic parsing each text into a formal knowledge tree, which represents the rule structure of those sentences without redundant words.

Then we convert the tree into a sequence of numerical values and set them as input variables corresponding to their output labels. The approach that we used is Tree-to-string Alignment Template (TAT) [45]. The template $z =< \tilde{T}, \tilde{S}, \tilde{A} >$ is a triple representation, which describes the alignment $\tilde{A}$ between the source parse tree $\tilde{T}$ and a target string $\tilde{S}$. The source parse tree is $\tilde{T} = T(F_1^{J'}$ is a sequence of leaf nodes, which contains both source words and phrasal categories. The target string $\tilde{S} = E_1^{I'}$ is a word string, which contains both target words and placeholders. An alignment $\tilde{A} \subseteq \{(j,i) : j = 1, ..., J'; i = 1, ..., I'\}$ is a subset of the Cartesian Product of source and target symbol positions. The TAT-based translation model can be decomposed into four sub-models:

- Parse model: $P(T(f_1^J)|f_1^J)$
- Detachment model: $P(D|T(f_1^J), f_1^J) \equiv P(\tilde{S}_1^K|\tilde{T}_1^K)$, where the hidden variable D detaches the source parse tree $T(f_1^J)$ into a sequence of K subtrees $\tilde{T}_i^K$ with preorder traversal. Assume that each subtree $\tilde{T}_k$ produces a target string $\tilde{S}_k$.
- Selection model: $P(z|\tilde{T})$
- Application model: $P(\tilde{S}|z, \tilde{T})$

Thus, we introduce the tree-to-string alignment templates z into probabilistic dependencies to model $P(e_1^I|f_i^J)$. The deduction of the model based on the four sub-models is the following:

$$\begin{aligned}
P(e_1^I|f_1^J) &= \sum_{T(f_1^J)} P(e_1^I, T(f_1^J)|f_1^J) \\
&= \sum_{T(f_1^J)} P(T(f_1^J)|f_1^J) P(e_1^I|T(f_1^J), f_1^J) \\
P(e_1^I|T(f_1^J), f_1^J) &= \sum_D P(e_1^I, D|T(f_1^J), f_1^J) \\
&= \sum_D P(D|T(f_1^J), f_1^J)(e_1^I|D, T(f_1^J), f_1^J) \\
&= \sum_D P(D|T(f_1^J), f_1^J) P(\tilde{S}_1^K|\tilde{T}_1^K) \\
&= \sum_D P(D|T(f_1^J), f_1^J) \prod_{k=1}^K P(\tilde{S}_1^K|\tilde{T}_1^K) \\
P(\tilde{S}|\tilde{T}) &= \sum_z P(\tilde{S}, z|\tilde{T}) \\
&= \sum_z P(z|\tilde{T}) P(\tilde{S}|z, \tilde{T})
\end{aligned}$$

The hidden variable $T(f_1^J)$ is omitted, because we only use the best parsing output. The hidden variable D is omitted, because we assume that all detachment have the same probability. Thus the model is simplified by parse, detachment, and sub-models.

$$P(e_1^I, z_1^K|f_1^J) = \frac{\exp[\sum_{m=1}^M \lambda_m h_m(e_1^I, f_1^J, z_1^K)]}{\sum_{e_1'^I, z_1'^K} \exp[\sum_{m=1}^M \lambda_m h_m(e_1'^I, f_1^J, z_1'^K)]}$$

$$\hat{e}_1^I = \underset{e_1^I, z_1^K}{\operatorname{argmax}} \sum_{m=1}^M \lambda_m h_m(e_1^I, f_1^J, z_1^K)$$

Feature functions analogous default feature set of Pharaoh[**koen-2004-pharaoh**] are:

$$h_1(e_1^I, f_1^J, z_1^K) = \log \prod_{k=1}^{K} \frac{N(z)\dot{\epsilon}(T(z), \tilde{T}_k)}{N(T(z))}$$

$$h_2(e_1^I, f_1^J, z_1^K) = \log \prod_{k=1}^{K} \frac{N(z)\dot{\epsilon}(T(z), \tilde{T}_k)}{N(S(z))}$$

$$h_3(e_1^I, f_1^J, z_1^K) = \log \prod_{k=1}^{K} lex(T(z)|S(z))\dot{\epsilon}(T(z), \tilde{T}_k)$$

$$h_4(e_1^I, f_1^J, z_1^K) = \log \prod_{k=1}^{K} lex(S(z)|T(z))\dot{\epsilon}(T(z), \tilde{T}_k)$$

$$h_5(e_1^I, f_1^J, z_1^K) = K$$

$$h_6(e_1^I, f_1^J, z_1^K) = \log \prod_{i=1}^{I} p(e_i|e_{i-2}, e_{i-1})$$

$$h_7(e_1^I, f_1^J, z_1^K) = I$$

6.4 Datasets

The dataset we used is WIKITABLEQUESTIONS [59]. It is a set of question answer pairs on HTML tables. The dataset is constructed by Wikipedia with at least 8 rows and 5 columns. Let one group of people to ask questions based on the table using 36 given generic prompts. Next, let the other group of people answer those questions based on the table. After completing the two tasks, only the answers agreed by at least two workers are kept in WIKITABLEQUESTIONS.

The dataset contains 22,033 examples on 2,108 tables. Those tables vary on topics and domains. This is a good choice for the base of Meta-Learning problem, since the input set of Meta-Learning should be a set of datasets. This is due to the mission of Meta-Learning is learning to learn in order to address the new encountered tasks or unforeseen tasks.

7 Conclusion

There are four ways to improve NLU.

- As mentioned in Meta-Learning Section, GPT-3 is facing methodological issues on natural language inference and reading comprehension. Thus it is need to improve logical reasoning ability on words that span long distances in text.
- The second direction in a narrower sense is to improve semantic parsing and grammar correction. Once getting a dataset of a specific task or domain, one needs to correct the grammar of each sample, so that the language model could be either pre-trained or fine-tuned more effectively and correctly. However, it is time consuming and low quality while correcting the grammar. To address this, we need to extend the domain adaptation of grammar correction. A domain-general grammar and lexicon [6] used in some applications had been designed, we need to further extend its domain. Alternatively we need to find an efficient way, e.g., hierarchical reinforcement learning, to correct the grammar.

- To make the performance of NLU closer to a human, only doing well in single and separate domain or task is insufficient. Because it is sometimes unable to find dataset of a specific domain, e.g., task-agnostic problem. Fortunately, GPT-3 has made a great progress to address this. Thus the third direction in a broader sense is either improving the performance of Meta-Learning on language model, or continuing to improve the performance of pre-training model.
- The fourth direction is that, if we see task-agnostic problem from a different angle, an alternative way of addressing this is to refer to information getting from speech recognition and image processing. Because those information is domain-free or task-free.

8 Future Work

I will continue to study on Bayesian Deep Learning, Topic Modeling, and Meta-Learning to seek for combination and improvement.

References

[1] Dimitris Achlioptas and Frank Mcsherry. *On Spectral Learning of Mixtures of Distributions*.

[2] Yoav Artzi. *Cornell SPF: Cornell Semantic Parsing Framework*. 2016. arXiv: 1311.3011 [cs.CL].

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[4] Bowen Baker et al. *Designing Neural Network Architectures using Reinforcement Learning*. 2017. arXiv: 1611.02167 [cs.LG].

[5] Jonathan Berant et al. "Semantic Parsing on Freebase from Question-Answer Pairs". In: *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1533–1544. URL: https://www.aclweb.org/anthology/D13-1160.

[6] *Broad-coverage Deep Language Understanding*. https://www.cs.rochester.edu/~james/. Jan.5 2021.

[7] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[8] S. Charles Brubaker and Santosh S. Vempala. *Isotropic PCA and Affine-Invariant Clustering*. 2008. arXiv: 0804.3575 [cs.LG].

[9] K. Chaudhuri and S. Rao. "Learning Mixtures of Product Distributions Using Correlations and Independence". In: *COLT*. 2008.

[10] Yutian Chen et al. *Learning to Learn without Gradient Descent by Gradient Descent*. 2017. arXiv: 1611.03824 [stat.ML].

[11] Rewon Child et al. *Generating Long Sequences with Sparse Transformers*. 2019. arXiv: 1904.10509 [cs.LG].

[12] Yulai Cong et al. "Deep latent Dirichlet allocation with topic-layer-adaptive stochastic gradient Riemannian MCMC". In: *arXiv preprint arXiv:1706.01724* (2017).

[13] S. Dasgupta and L. Schulman. “A Two-Round Variant of EM for Gaussian Mixtures”. In: *ArXiv* abs/1301.3850 (2000).

[14] Sanjoy Dasgupta. “Learning Mixtures of Gaussians”. In: *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. FOCS ’99. USA: IEEE Computer Society, 1999, p. 634. ISBN: 0769504094.

[15] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[16] Li Dong and Mirella Lapata. *Language to Logical Form with Neural Attention*. 2016. arXiv: 1601.01280 [cs.CL].

[17] Yan Duan et al. *RL^2: Fast Reinforcement Learning via Slow Reinforcement Learning*. 2016. arXiv: 1611.02779 [cs.AI].

[18] Benjamin Eysenbach et al. *Diversity is All You Need: Learning Skills without a Reward Function*. 2018. arXiv: 1802.06070 [cs.AI].

[19] Xing Fan et al. *Transfer Learning for Neural Semantic Parsing*. 2017. arXiv: 1706.04326 [cs.CL].

[20] Chelsea Finn, Pieter Abbeel, and Sergey Levine. *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. arXiv:1703.03400 [cs.LG]. 2017. arXiv: 1703.03400 [cs.LG].

[21] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *CoRR* abs/1703.03400 (2017). arXiv: 1703.03400. URL: http://arxiv.org/abs/1703.03400.

[22] Zhe Gan et al. “Scalable deep Poisson factor analysis for topic modeling”. In: *International Conference on Machine Learning*. 2015, pp. 1823–1832.

[23] Alex Graves, Greg Wayne, and Ivo Danihelka. *Neural Turing Machines*. 2014. arXiv: 1410.5401 [cs.NE].

[24] Aditya Grover, Aaron Zweig, and Stefano Ermon. “Graphite: Iterative generative modeling of graphs”. In: *International conference on machine learning*. PMLR. 2019, pp. 2434–2444.

[25] Bharath Hariharan and Ross Girshick. *Low-shot Visual Recognition by Shrinking and Hallucinating Features*. 2017. arXiv: 1606.02819 [cs.CV].

[26] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[27] Charles T. Hemphill, John J. Godfrey, and George R. Doddington. “The ATIS Spoken Language Systems Pilot Corpus”. In: *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27,1990*. 1990. URL: https://www.aclweb.org/anthology/H90-1021.

[28] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV].

[29] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[30] Hengguan Huang et al. *Deep Graph Random Process for Relational-Thinking-Based Speech Recognition*. 2020. arXiv: 2007.02126 [cs.LG].

[31] Forrest N. Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size*. 2016. arXiv: 1602.07360 [cs.CV].

[32] Ravindran Kannan, Hadi Salmasian, and Santosh Vempala. "The Spectral Method for General Mixture Models". In: *Proceedings of the 18th Annual Conference on Learning Theory*. COLT'05. Bertinoro, Italy: Springer-Verlag, 2005, pp. 444–457. ISBN: 3540265562. DOI: 10.1007/11503415_30. URL: https://doi.org/10.1007/11503415_30.

[33] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG].

[34] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[35] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: 2001.04451 [cs.LG].

[36] Gregory R. Koch. "Siamese Neural Networks for One-Shot Image Recognition". In: 2015.

[37] Gregory Kuhlmann, Peter Stone, and Raymond Mooney. "Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer". In: *In Proc. of the AAAI-04 Workshop on Supervisory Control of Learning and Adaptive Systems*. 2004.

[38] Hao Li et al. *Pruning Filters for Efficient ConvNets*. 2017. arXiv: 1608.08710 [cs.CV].

[39] Ke Li and Jitendra Malik. *Learning to Optimize Neural Nets*. 2017. arXiv: 1703.00441 [cs.LG].

[40] Sheng Li, Jaya Kawale, and Yun Fu. "Deep Collaborative Filtering via Marginalized Denoising Auto-Encoder". In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM '15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 811–820. ISBN: 9781450337946. DOI: 10.1145/2806416.2806527. URL: https://doi.org/10.1145/2806416.2806527.

[41] Xiaopeng Li and James She. "Collaborative variational autoencoder for recommender systems". In: *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017, pp. 305–314.

[42] Percy Liang. *Lambda Dependency-Based Compositional Semantics*. 2013. arXiv: 1309.4408 [cs.AI].

[43] Yi Liao et al. "Quase: Sequence editing under quantifiable guidance". In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2018, pp. 3855–3864.

[44] Wang Ling et al. *Latent Predictor Networks for Code Generation*. 2016. arXiv: 1603.06744 [cs.CL].

[45] Yang Liu, Qun Liu, and Shouxun Lin. "Tree-to-string alignment template for statistical machine translation". In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*. 2006, pp. 609–616.

[46] David JC MacKay. "A practical Bayesian framework for backpropagation networks". In: *Neural computation* 4.3 (1992), pp. 448–472.

[47] Dougal Maclaurin, David Duvenaud, and Ryan P. Adams. *Gradient-based Hyperparameter Optimization through Reversible Learning*. 2015. arXiv: 1502.03492 [stat.ML].

[48] Diego Marcheggiani, Anton Frolov, and Ivan Titov. *A Simple and Accurate Syntax-Agnostic Neural Model for Dependency-based Semantic Role Labeling*. 2017. arXiv: 1701.02593 [cs.CL].

[49] Nikhil Mehta, Lawrence Carin, and Piyush Rai. "Stochastic blockmodels meet graph neural networks". In: *arXiv preprint arXiv:1905.05738* (2019).

[50] Stephen Merity. *Single Headed Attention RNN: Stop Thinking With Your Head*. 2019. arXiv: 1911.11423 [cs.CL].

[51] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. "Regularizing and Optimizing LSTM Language Models". In: *International Conference on Learning Representations*. 2018. URL: https://openreview.net/forum?id=SyyGPP0TZ.

[52] *Microsoft Research Blog*. https://www.microsoft.com/en-us/research/blog/. Feb 2020.

[53] Jonas Mueller, David Gifford, and Tommi Jaakkola. "Sequence to better sequence: continuous revision of combinatorial structures". In: *International Conference on Machine Learning*. 2017, pp. 2536–2544.

[54] Tsendsuren Munkhdalai and Hong Yu. *Meta Networks*. 2017. arXiv: 1703.00837 [cs.LG].

[55] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN: 0262018020.

[56] Renato Negrinho and Geoff Gordon. *DeepArchitect: Automatically Designing and Training Deep Architectures*. 2017. arXiv: 1704.08792 [stat.ML].

[57] Alex Nichol, Joshua Achiam, and John Schulman. *On First-Order Meta-Learning Algorithms*. arXiv:1803.02999 [cs.LG]. 2018. arXiv: 1803.02999 [cs.LG].

[58] Y. Oda et al. "Learning to Generate Pseudo-Code from Source Code Using Statistical Machine Translation". In: *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2015, pp. 574–584. DOI: 10.1109/ASE.2015.36.

[59] Panupong Pasupat and Percy Liang. "Compositional Semantic Parsing on Semi-Structured Tables". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1470–1480. DOI: 10.3115/v1/P15-1142. URL: https://www.aclweb.org/anthology/P15-1142.

[60] Hao Peng, Sam Thomson, and Noah A. Smith. *Deep Multitask Learning for Semantic Dependency Parsing*. 2017. arXiv: 1704.06855 [cs.CL].

[61] Chris Quirk, Raymond Mooney, and Michel Galley. "Language to Code: Learning Semantic Parsers for If-This-Then-That Recipes". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 878–888. DOI: 10.3115/v1/P15-1085. URL: https://www.aclweb.org/anthology/P15-1085.

[62] A. Radford. "Improving Language Understanding by Generative Pre-Training". In: 2018.

[63] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[64] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG].

[65] Sachin Ravi and Hugo Larochelle. "Optimization as a Model for Few-Shot Learning". In: *5th International Conference on Learning Representations, (ICLR) 2017*. Toulon, France, April 24-26, 2017: OpenReview.net, 2017. URL: https://openreview.net/forum?id=rJY0-Kcll.

[66] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[67] Arora Sanjeev and Ravi Kannan. "Learning Mixtures of Arbitrary Gaussians". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 247–257. ISBN: 1581133499. DOI: 10.1145/380752.380808. URL: https://doi.org/10.1145/380752.380808.

[68] Adam Santoro et al. "Meta-Learning with Memory-Augmented Neural Networks". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1842–1850.

[69] Mohammad Shoeybi et al. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. 2020. arXiv: 1909.08053 [cs.CL].

[70] Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical Networks for Few-Shot Learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 4080–4090. ISBN: 9781510860964.

[71] Flood Sung et al. *Learning to Compare: Relation Network for Few-Shot Learning*. 2018. arXiv: 1711.06025 [cs.CV].

[72] Ivan Titov and Alexandre Klementiev. "A Bayesian Model for Unsupervised Semantic Parsing". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, June 2011, pp. 1445–1455. URL: https://www.aclweb.org/anthology/P11-1145.

[73] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].

[74] S. Vempala and Grant Wang. "A spectral algorithm for learning mixtures of distributions". In: *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.* (2002), pp. 113–122.

[75] Oriol Vinyals et al. *Matching Networks for One Shot Learning*. 2017. arXiv: 1606.04080 [cs.LG].

[76] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. "Collaborative recurrent autoencoder: Recommend while learning to fill in the blanks". In: *Advances in Neural Information Processing Systems* 29 (2016), pp. 415–423.

[77] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. *Natural-Parameter Networks: A Class of Probabilistic Neural Networks*. 2016. arXiv: 1611.00448 [cs.LG].

[78] Hao Wang, Xingjian Shi, and Dit-Yan Yeung. "Relational deep learning: A deep latent variable model for link prediction". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[79] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. *Collaborative Deep Learning for Recommender Systems*. 2015. arXiv: 1409.2944 [cs.LG].

[80] Hao Wang and Dit-Yan Yeung. *Towards Bayesian Deep Learning: A Framework and Some Existing Methods*. 2016. arXiv: 1608.06884 [stat.ML].

[81] Hao Wang and Dit-Yan Yeung. *A Survey on Bayesian Deep Learning*. 2020. arXiv: 1604.01662 [stat.ML].

[82] Hao Wang et al. *Bidirectional Inference Networks: A Class of Deep Bayesian Networks for Health Profiling*. 2019. arXiv: 1902.02037 [stat.ML].

[83] Jane X Wang et al. *Learning to reinforcement learn*. 2017. arXiv: 1611.05763 [cs.LG].

[84] Yushi Wang, Jonathan Berant, and Percy Liang. "Building a Semantic Parser Overnight". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1332–1342. DOI: 10.3115/v1/P15-1129. URL: https://www.aclweb.org/anthology/P15-1129.

[85] Jason Weston, Sumit Chopra, and Antoine Bordes. *Memory Networks*. 2015. arXiv: 1410.3916 [cs.AI].

[86] Yuk Wah Wong and Raymond J. Mooney. "Learning for Semantic Parsing with Statistical Machine Translation". In: *Proceedings of Human Language Technology Conference / North American Chapter of the Association for Computational Linguistics Annual Meeting (HLT-NAACL-06)*. New York City, NY, 2006, pp. 439–446. URL: http://www.cs.utexas.edu/users/ai-lab?wong:naacl06.

[87] Liang Yao, Chengsheng Mao, and Yuan Luo. *Graph Convolutional Networks for Text Classification*. 2018. arXiv: 1809.05679 [cs.CL].

[88] Haochao Ying et al. "Collaborative deep ranking: A hybrid pair-wise recommendation algorithm with implicit feedback". In: *Pacific-asia conference on knowledge discovery and data mining*. Springer. 2016, pp. 555–567.

[89] John M. Zelle and Raymond J. Mooney. "Learning to Parse Database Queries Using Inductive Logic Programming". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*. AAAI'96. Portland, Oregon: AAAI Press, 1996, pp. 1050–1055. ISBN: 026251091X.

[90] Fuzheng Zhang et al. "Collaborative knowledge base embedding for recommender systems". In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 353–362.

[91] Han Zhang et al. *Self-Attention Generative Adversarial Networks*. 2019. arXiv: 1805.08318 [stat.ML].

[92] He Zhao et al. "Dirichlet belief networks for topic structure learning". In: *Advances in neural information processing systems*. 2018, pp. 7955–7966.

[93] Barret Zoph and Quoc V. Le. *Neural Architecture Search with Reinforcement Learning*. 2017. arXiv: 1611.01578 [cs.LG].