Using Sparse Parameter Estimation for Semantic Parsing

Jiayu Zhou, Jieping Ye, Juraj Dzifcak, Chitta Baral

Arizona State University

Abstract. This paper addresses the problem of semantic parsing, by which natural language sentences are translated into a form which conveys their underlying meaning. Semantic parsing involves a parameter estimation process, which is a convex optimization problem. The optimization formulation of previous approaches often requires huge amount of time to converge due to the high dimensional feature space. In this paper we introduce a fast semantic parsing framework which uses ℓ_1-norm regularized learning to get a sparse model and better convergence speed. Experiments demonstrate overall higher performance of our semantic parsing system using inverse λ, generalization and ℓ_1 regularization. Regularized parameter updating shows significantly improvement on the learning speed and reduced model size.

1 Introduction

In the field of natural language processing, semantic parsing refers to the process by which natural language sentences are translated into a form which conveys their underlying meaning, as opposed to traditional parsing which translates a sentence into a representation of its syntactic structure. The underlying meaning representation can be any kind of knowledge representation e.g. *first-order logic*, *database query languages*, *temporal logic*, and *answer set programming (ASP)*[1]. In the broader field of artificial intelligence, semantic parsing has a considerable advantage over traditional parsing in that it allows for high level reasoning techniques to be applied, producing non-trivial inferences from the input text.

Intensive efforts are made in the area of semantic parsing, learning from pairs of natural language (NL) sentence and corresponding semantic representation [3–9, 11]. Given a set of training samples of natural language sentences and corresponding logical forms, these approaches build models which are capable of translating unseen NL sentences to their correct logic forms. Recently it is also shown that such frameworks, as proposed in [6] and others, do not limit to a specific kind of NL [2] and can be used for multilingual purposes.

It is noteworthy that in the lexicon building process many works have included handcrafted entries or rules, such as [5, 6]. In this way different sets of rules need to be built for different target languages, which restricts the target languages that are applicable. In our paper we use an approach that makes use of an existing syntactic parser and infers semantics of unknown words from known

ones, and the learnt semantics can be generalized to other unseen words of the same syntactic category. This method is expected to learn the semantics that cannot be easily covered by handcrafted rules and target languages are no longer limited in the way that previous works did. Note that in this paper we focus on the parameter estimation and therefore definitions and algorithms of inverse λ and generalization are given without proof. Details of inverse and generalization and corresponding proofs will be elaborated in a separate paper[24].

In the parameter estimation parameter part, most existing works use maximum likelihood method [2, 3, 5, 6] or its similar dual form – the maximum entropy method [7–9] to train on a log-linear model or conditional random field (CRF). All these methods are readily solved by gradient descent methods because the loss function formulated is a smooth convex problem.[1] In our proposed system we add the ℓ_1-norm regularization to the loss function in order to introduce sparsity to the model. Because the ℓ_1-norm is not differentiable at zero, and therefore gradient methods cannot be directly used. To solve the non-smooth part, we use the modified sub-gradient approach illustrated in Sec. 4, which gives more reliable sparsity than directly use of sub-gradient.

We evaluate our system on two benchmark datasets: GeoQuery and CLANG. GeoQuery is a set of 880 queries to a database of United States geography and their corresponding English sentences. CLANG is a set of 600 rules to used for directing simulated soccer game. We compare performance to previous methods[4, 5, 11, 9, 8, 7] in terms of precision and recall. To further investigate the effect of sparsity, we evaluate how our model works under different ℓ_1 regularization factor. We also visualize such sparsity in a small feature space showing how a compact model generates in the training process.

2 Background

In our system semantics of words are represented by typed lambda calculus[13]. One advantage of using the lambda calculus is that the final logical form of a sentence can take any types of semantics. Combinatory Categorical Grammar (GGC)[14], a parsing formalism that tightly couples syntax and semantics, is used to direct the application of semantic of words to get the semantic representation of sentences. The CCG grammar includes a lexicon $\Lambda = \{(w, cat, sem)\}$, where w is a word, cat is its syntactic category, and sem is the semantics of the word. Methods described in [6, 5] are used to parse a sentence and give its semantic representation.

Because of meaning and structural ambiguity, thus provided a sentence, there may be multiple semantic representations in the parsing result. Probabilistic CCG (PCCG) is proposed for the disambiguation purpose and provides probabilistic distributions of translated logical forms. Given a lexicon Λ, we are interested in the probabilistic distribution of a particular logical representation

[1] The originally formulated likelihood function $\mathcal{L}$ is a concave problem and gradient ascent methods are actually used. Here we are using convex because we try to keep on the convention of most optimization literatures.

y provided the natural language sentence x. We refer z as the parsing tree of x that generates the logical representation y, the PCCG defines a conditional distribution $P(x, z|y)$ of (x, z) for a given sentence y.

In our system we employ the conditional random field (CRF) or log-linear model for CCGs introduced by Clark & Curran [16]. We denote ϕ as our feature function that maps the tuple (x, z, y) to $\mathbb{R}^d$, where d is the number of features we use in the model. The model is parameterized by a weight vector $\theta \in \mathbb{R}^d$. The desired conditional distribution is given by

$$P(x, z|y; \theta, \Lambda) = \frac{e^{\theta \cdot \phi(x,z,y)}}{\sum_{x',z'} e^{\theta \cdot \phi(x',z',y)}} \tag{1}$$

In the decision stage, the task of the model is to find the distribution of logical representation y that has the highest probability:

$$arg\max_{y} P(y|x; \Lambda) = arg\max_{y} \sum_{z} P(y, z|x; \Lambda) \tag{2}$$

where z can be considered as a latent variable and is factored out. In our implementation we use a dynamic programming algorithm that is similar to CKY algorithm to calculate the probability.

In the estimation stage, we are given a set of training pairs $\{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$, where n is the sample size. Given the parameter vector θ, the likelihood of each pair is given by $\mathcal{L}_i(\theta, \Lambda) = \log P(y_i|x_i; \theta, \Lambda)$. The object of training process is to maximize the log-likelihood of the entire training set given by:

$$\mathcal{L}(\theta, \Lambda) = \sum_{i=1}^{n} \log P(y_i|x_i; \theta, \Lambda) = \sum_{i=1}^{n} \log \sum_{z'} P(y_i, z'|x_i; \theta, \Lambda) \tag{3}$$

$$= \sum_{i=1}^{n} \log \sum_{z'} e^{\phi(y_i, z'|x_i; \theta, \Lambda) \cdot \theta} - \sum_{i=1}^{n} \log \sum_{z', y'} e^{\phi(y', z'|x_i; \theta, \Lambda) \cdot \theta} \tag{4}$$

In order to obtain the optimal value θ that maximize $\mathcal{L}$, we set its partial derivative with respect to each dimension of θ to zero and get following equation:

$$\frac{\partial \mathcal{L}(\theta, \Lambda)}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}_i(\theta, \Lambda)}{\partial \theta_j} = \sum_{i=1}^{n} \sum_{z'} P(y'|x_i, y_i; \theta, \Lambda) \cdot f_j(y_i, z', x_i) \tag{5}$$

$$- \sum_{i=1}^{n} \sum_{y', z'} P(y', z'|x_i; \theta, \Lambda) \cdot f_j(y', z', x_i) \tag{6}$$

Because there is no analytical solution if we set above equation to zero, gradient methods are readily used in previous researches to solve this iteratively[17]. The basic idea of stochastic gradient ascent that is widely used in previous works is that perform

$$\theta_j = \theta_j + \alpha(t) \frac{\partial \mathcal{L}_i(\theta, \Lambda)}{\partial \theta_j} = \theta_j + \alpha(t) \frac{\partial \log P(y_i|x_i; \theta, \Lambda)}{\partial \theta_j} \tag{7}$$

to each weight θ_i until some convergence conditions are reached, where $\alpha(t)$ is the step size, a function of number of total previous update times t.

3 Lexicon Learning

Lexicon acquisition is an important component of semantic parsing. Handcrafted lexicon is only possible for systems with very limited vocabulary, and therefore automatic lexicon learning methods are proposed [6, 5]. The proposed methods, however, are heavily dependent on a table of rules and thus restrict the types of semantics that can be extracted. In our system we use inverse λ algorithm and generalization in order to break such constraints[2].

3.1 The Inverse λ Operators

The inverse algorithm includes two inverse λ operators: INVERSE-L and INVERSE-R, which compute typed λ-calculus F given H and G such that $F@G = H$ and $G@F = H$ respectively. In the following context we use J^i to denote typed terms. J_i are used to denotes sub terms of J, v_i, v and w denote variables, σ_i represents atomic terms. $f(\cdot)$ represents a typed atomic formula. Note that the list of λ-abstractors of the form $\lambda v_1, \ldots, v_i$ can be empty.

Definition 1 (Operator :). *Consider two lists of typed λ-elements $A = (a_1, \ldots, a_m)$ and $B = (a_1, \ldots, a_n)$ and a formula H. The result of the operation $H(A : B)$ takes every typed λ-element a_i from H and is substituted by the corresponding typed λ-element b_i of the list B until all elements have been substituted following the order of the list, for each appearance of A in H.*

Based on the definition the INVERSE-R function can be presented as:
Procedure INVERSE-R(H, G)
if G is $\lambda v.v@J$
then $F =$ INVERSE-L(H, J)
if J is a sub term of H and G is $\lambda v.H(J : v)$
then $F = J$
if G is not $\lambda v.v@J$, J is a sub term of H and G is $\lambda w.\quad H(J(J_1, \ldots, J_m) : w@J_p, \ldots, @J_q)$, where $1 \leq p, q, s \leq m$.
then $F = \lambda v_1, \ldots, v_s.J(J_1, \ldots, J_m : v_p, \ldots, v_q)$.
else $F =$NULL
Procedure INVERSE-L(H, G) is defined similarly.

One disadvantage of inverse operators is that they cannot be used when there are more than one word with unknown semantics in the sentence. In such cases we allow a trivial solution in which $\lambda x.x$ is assigned as the semantic to the unknown word under certain conditions. In our system any word not presenting in the final semantics is a potential candidate to be assigned the trivial semantic representation $\lambda x.x$, such as word "the". Once a non-trivial one is found, however, the system attempts to use it over the trivial one. Comparing to lexicon entries generated by inverse, the trivial lexicon entries have much lower initial weights.

[2] Please note that details of inverse-λ and generalization are elaborated in another paper and are not new contribution of this paper

3.2 Generalization

Inverse operators are able to obtain new semantic representation for particular words in sentences. And we want to further extend the learned semantics to words that are outside the training data. Given a word w with category cat, a simple solution is to find some semantics of other words of the same category, and build semantics for w based on that word. For example, suppose in the lexicon we already have a lexicon entry fly of category $S\backslash NP$ with semantic $\lambda x.fly(x)$. For each word w of the same category, the generalization function *GENERALIZE* generates semantic expression $\lambda.w(x)$ and adds it to the lexicon. This idea, however, has an obvious disadvantage that it generates large amount of new lexicon entries,and clearly many of them will not be useful. Therefore we only perform generalization whenever a sentence contains words with unknown semantics.

We now present the generalization algorithm GENERALIZE(Λ, α), where Λ is our lexicon and α is the word with unknown semantics. Denoting each lexicon entry as triple $l = (w, cat, sem)$, we can use $l(w)$,$l(cat)$ and $l(sem)$ to represent the word string, category and semantics of the entry respectively. The algorithm of GENERALIZE(Λ, α) contains two sub-procedures. IDENTIFY(w, sem) identifies the parts of sem in which w is involved and REPLACE(sem, a, b) replaces a with b in sem.

Procedure GENERALIZE(Λ, α)
  for $l \in \Lambda$
  if $l(cat) = \alpha(cat)$
    I = IDENTIFY($l(w), l(sem)$)
    sem' = REPLACE ($l(sem), I, \alpha(w)$)
    $\Lambda = \Lambda \cup (\alpha(w), \alpha(cat), sem')$

4 Sparse Parameter Estimation

While using inverse λ and generalization helps us to find correct meanings of unknown words, it also introduces many irrelevant entries to the dictionary, which will never be used in the correct logic representation of any sentence. These irrelevant entries, however, are used as lexical features and thus tremendously increase time of parsing and also the converge time of parameter estimation. This motivates us to introduce sparse to get a more compact model.

Recall that in our parameter estimation stage we are maximizing the objective function $\mathcal{L}$ with respect to θ. In typical situations, however, direct training the model is highly likely to cause overfitting problems [18], which means the parameters are tuned to perform very good on the training data but have poor predictive performance on the test data. To solve the problem, a regularization term $\mathcal{R}(\theta)$ is usually introduced in the objective function that penalizes θ from being over-trained, and now we need to solve the optimization problem:

$$arg\max_{\theta} \mathcal{O}(\theta) = arg\max_{\theta}(\mathcal{L}(\theta, \Lambda) - \mathcal{R}(\theta)) \tag{8}$$

The most common regularization functions are the $\ell_1 = \|\theta\|_2^2$ and $\ell_2 = \|\theta\|_1$ regularization. While using ℓ_2 norm has the obvious advantage over ℓ_1 resulting a smooth term, ℓ_1 norm regularization showed theoretically and empirically that it yields models in which most of the features are irrelevant[19], i.e. it results in sparse a model that we expect. The formulation then has the form:

$$arg\max_{\theta}(\mathcal{L}(\theta, \Lambda) - \beta\|\theta\|_1) \tag{9}$$

where β is the parameter to control the degree of regularization. ℓ_1 regularization has a significant disadvantage however in that it is not differentiable at zero, and as a consequence, the gradient-based optimization algorithm proposed in (7) cannot be directly used. Using gradient-based algorithm on (9), we can get following update

$$\theta_i^{k+1} = \theta_i^k + \alpha(t)\frac{\partial\mathcal{L}_j(\theta, \Lambda) - \frac{\beta}{n} \mid \theta_i \mid}{\partial\theta} \tag{10}$$

where N denotes number of iterations, $\mathcal{L}_j(\theta, \Lambda) = \log P(y_j|x_j; \theta, \Lambda)$ is the likelihood given jth training sample, θ_i^k is the value of θ_i at iteration k, n is number of training samples. The simplest remedy of the non-smooth problem is to consider a subgradient at zero and take advantage of the sign function $\sigma(x)$ as follows:

$$\theta_i^{k+1} = \theta_i^k + \alpha(t)\frac{\partial\mathcal{L}_j(\theta, \Lambda)}{\partial\theta} - \frac{\beta}{n}\alpha(t)\sigma(\theta_i) \tag{11}$$

where $\sigma(x) = 1$ if $x > 0$, $\sigma(x) = -1$ if $x < 0$ and $\sigma(x) = 0$ if $x = 0$. However, this algorithm leads to a series of problems and cannot guarantee that we get a compact model. Many approaches are proposed to solve the problem [20, 21]. In our system we adopt the ℓ_1 regularization with cumulative penalty, a two-step update process, whose main idea is to smooth out the effect of fluctuating gradients by considering the cumulative effects from ℓ_1 penalty[21].

The cumulative penalty method uses a two-step update strategy for each iteration. First, we update the parameters with the derivative of the smooth part, as we do in the unregularized formulation. We denote the result by $\theta_i^{t+\frac{1}{2}}$. Then the cumulative penalty is applied on the update, denoted by θ_i^{t+1}. Let u_k be the absolute value of the total ℓ_1 penalty that each weight could have receive up to the point. This can be accumulated as $u_k = \frac{\beta}{n}\sum_{t=1}^{k}\alpha(t)$. Let q_i^k be the total ℓ_1 penalty that θ_i has actually received up to the current iteration: $q_i^k = \sum_{t=1}^{k}(\theta_i^{t+1} - \theta_i^{t+\frac{1}{2}})$. The second update step can be presented in the following way:

$$\theta_i^{k+\frac{1}{2}} = \theta_i^k + \alpha(t)\frac{\partial L_j(\theta, \Lambda)}{\partial\theta_i}$$

$$\textbf{if} \quad \theta_i^{(k+\frac{1}{2})} > 0 \quad \textbf{then} \qquad \theta_i^{(k+1)} = \max(0, \theta_i^{(k+\frac{1}{2})} - (u_k + q_i^k))$$

$$\textbf{else if} \quad \theta_i^{(k+\frac{1}{2})} < 0 \quad \textbf{then} \qquad \theta_i^{(k+1)} = \min(0, \theta_i^{(k+\frac{1}{2})} + (u_k - q_i^k))$$

We denote call two-step parameter update procedure SPARSE-UPDATE(θ, Λ), which gives the updated θ as output.

Procedure SPARSE-UPDATE(θ, Λ)
  for $\theta_i \in$ features used in sample j
  $\theta_i = \theta_i + \alpha(t)\frac{\partial \mathcal{L}_j(\Lambda,\theta)}{\partial \theta_i}$
  APPLY-PENTALTY(θ_i)
Procedure APPLY-PENALTY(i)
  $z = \theta_i$
  if $\theta_i > 0$ **then** $\theta_i = \max(0, \theta_i - (u + q_i))$
  else if $\theta_i < 0$ **then** $\theta_i = \min(0, \theta_i + (u - q_i))$
  $q_i = q_i + (\theta_i - z)$

5 Overall Learning Algorithm

Fig. 5 presents our fast semantic parsing system using inverse λ and generalization. For each iteration there are two steps. First, new lexical entries are learned using inverse algorithm and generalization, we get the new lexicon Λ^{t+1}. Then the PCCG parameter θ^t is updated by our sparse stochastic gradient method, given the updated lexicon. By factor training samples out from the iterations, this model can be trivially turned to online fashion.

Inputs and Initialization. The algorithm takes a training samples (NL sentence, logical form) of n: $S = \{(x_i, y_i) : i = 1 \dots n\}$, an initial lexicon Λ_0 and a parameter vector θ_0 whose elements are set to 0.01. We use lexical feature as described in [6].

Step 1: Lexicon Update. In the lexicon update step the algorithm iterates over all sentences n times. For each sentence, using the CCG parser to get its corresponding parse tree. Inverse λ algorithm is then performed on the tree to learn unknown lexicon entries. All lexicon entries learnt are generalized and stored in the lexicon.

Step 2: Sparse Parameter Update. For each training sample we update the parameter θ using the stochastic gradient ascent method on the ℓ_1-norm regularized log-linear model, as discussed in Sec.4.

6 Experiment

In this section we conduct three kinds of experiments to evaluate different aspects of the system. The first experiment compares our proposed system to the existing systems, this shows how our system performs in terms of precision and recall. Next, different sparsity settings (β) of model are evaluated independently within our system, and we are able to inspect the relationship among precision, recall, and sparsity. We want to impose moderate sparsity, by which we can quickly obtain a model that is considerably compressed, without compromising much on precision and recall. Finally we use a small lexicon to visualize how the sparsity is produced in the feature space during the parameter estimation process.

We use only lexical features in our system, instead of other structure features such as syntactic features and context features, because ℓ_1-norm does not consider internal structures within the features. In the first two experiments we used

Input

$S = \{(x_i, y_i) : i = 1 \dots n\}$: Training samples, x_i are sentences and y_i are its corresponding expressions.

Λ^0 Initial lexicon.

θ^0 Initial feature weights.

T Training iterations.

Algorithm

- FOR $t = 1 \dots T$
- **Step 1:** (Lexical Generalization)
- FOR $i = 1 \dots n$
  - FOR $j = 1 \dots n$
  - Parse x_j to get its parse tree z_j
  - Traverse z_j and apply INVERSE-L, INVERSE-R and GENERALIZE to find new λ-calculus expressions of words and phrases α
  - Set $\Lambda^{t+1} = \Lambda^t \cup \alpha$
- **Step 2:** (Sparse Parameter Estimation)
- Set θ^{t+1} = SPARSE-UPDATE $(\theta^t, \Lambda^{t+1})$

Output

Λ^{T+1}: An update lexicon

θ^{T+1}: An update feature weights.

Fig. 1. Overall learning framework of fast semantic parsing system using inverse λ and generalization.

10 fold cross validation on both datasets. The syntactic parser we use is the *C&C* parser from Clark and Curran[16]. *Precision* and *recall* are two measurements we use. Here precision is the percentage of returned semantic representation that are correct and recall is the percentage of test data with correct logical semantic representation parsed. Also we use *F1-measure*, which is calculated in the way that precision and recall are equally considered. For the convergence criteria we use the difference of objective function value $\Delta\mathcal{O}(\theta) \leq 10^{-4}$.

In the first experiment we compare across our system and previous systems on the GeoQuery and CLANG respectively. INV+ is our semantic parsing framework with trivial inverse and generalization, where ℓ_1 regularization is disabled ($\beta = 0$). Disabling the regularization means the parameter estimation of INV+ is equivalent to the one used in [6, 5]. For SPINV+ the sparsity $\beta = 0.3$ is used. Other systems involved in our comparison are:GOLDSYN[7], WASP[8], SCISSOR[10], KRISP[11], Semantic Parser by Zettlemoyer and Collins[5], and one in the work by Lu et. al[4]. Result of this part is shown in Table 1.

From the result we see that INV+ and SPINV+ outperformed all other systems in both datasets as we expected. Moderate sparsity as we use in SPINV+ only slightly reduces F-measure on GeoQuery, and produces even better results on CLANG. Because of the large feature space the system produce from inverse and generalization (around 32400 lexical features for GeoQuery and 26500 for CLANG), overfit situations are likely to occur without proper regularization. The ℓ_1-norm regularization, on the other hand, performs a feature selection

System	Precision	Recall	F1
INV+	93.41	89.04	91.17
SPINV+	93.13	88.50	90.76
GOLDSYN	91.94	88.18	90.02
WASP	91.95	86.59	89.19
SCISSOR	95.50	77.20	85.38
KRISP	93.34	71.70	81.10
ZC05	91.63	86.07	88.76
LU08	89.30	81.50	85.20

GeoQuery

System	Precision	Recall	F1
INV+	85.74	76.63	80.92
SPINV+	86.33	78.24	82.09
GOLDSYN	84.73	74.00	79.00
WASP	88.85	61.93	72.99
SCISSOR	89.50	73.70	80.80
KRISP	85.20	61.85	71.67
LU08	82.50	67.70	74.40

CLANG

Table 1. Performance on GeoQuery and CLANG.

during the training process, as weights of some features are clipped to zero. This explains why sparsity works better in some situations.

For the parameter estimation, INV+ does not converge and algorithm stops when maximum iteration number 1000 is reached, while SPINV+ converges at 400 iterations. Also it is noticeable that SPINV+ uses only 20% features (with non-zero weights) as does in INV+. With comparable performance as INV+, SPINV+ strikingly reduces parameter estimation time and model size.

The next thing we are interested in from the ℓ_1-norm regularized model is what models of different compression rates can bring us. We define the *compression-ratio* to be $(||\theta||_0 - ||\theta'||_0)/(||\theta||_0)$. where $||\cdot||_0$ is zero norm, and is defined by the non-zero elements in the vector x, and θ' is the vector of weights after training. Intuitively, the compress-ration denotes how many features are considered to be irrelevant in comparison to the original feature space.

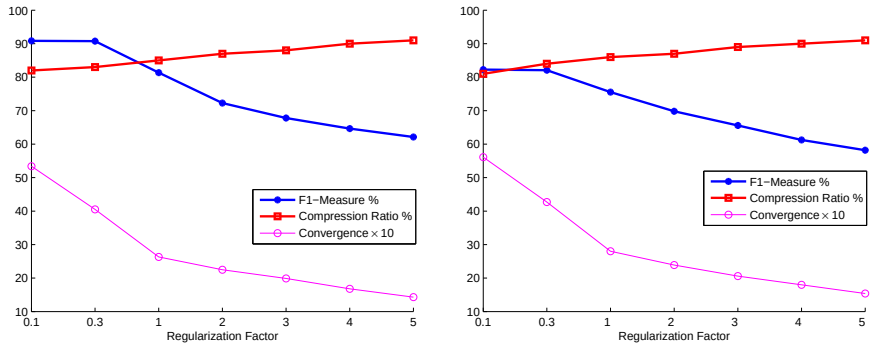

Fig. 2. Sparsity Performance Evaluation

The evaluation of the performance of sparsity is shown in Fig. 2, where the change of F1 measure, compression ratio, and convergence rate are plotted

against the change of regularization factor. The general pattern is: when the regularization factor increases, the F1-measure and convergence time decrease, and meanwhile compression ratio increases, as we expect. In GeoQuery, if we consider $\beta = 0.1$, the model has almost the same F1-measure as that of the unregularized model, while it compresses around 80% features, leaving less than 20% features with non-zero weights. This striking compression ratio enables us to deal with much larger domains that all previous works cannot handle. Also the exponential decreasing convergence time makes large-scale training corpus feasible. Similar patterns are found in the CLANG dataset. While applying semantic parsing in domain independent scenarios with huge size of feature space, ℓ_1 regularization allows us to choose a sparsity that balance among training speed, model size and performance.

Fig. 3 shows how the weights change using a small lexicon with a single natural language sentence, "John takes a plane", used as the training set. The first graph shows the evolution of the weights without regularization, while the second shows the evolution when ℓ_1-norm regularization is employed. Note that in our experimental setting, when new words are added to lexicon, they are assigned a default weight of 0.01.

Without regularization, during each iteration of the algorithm, only those weights for lexicon entries used in the current training sample are affected. Suppose that for a given iteration our training sample is "John takes a plane". In this iteration on the weights for the lexicon entries of the worlds: *john*, *take*, *a*, and *plane* will be affected. For this example, we have the correct category $john \mapsto john$, and an incorrect one: $john \mapsto \lambda\, x.x@mary$. With each iteration the weight for $john \mapsto john$ will increase, while the weight of $john \mapsto \lambda\, x.x@mary$ will decrease. It is highly probable that the weight for the incorrect category will fall below zero once the training process has finished. Utilizing ℓ_1-norm regularization however avoids this problem. It is evident from Fig. 3 that the weight for unused features approaches zero in the first several training iterations. The use of clipping however causes to not fall below zero.

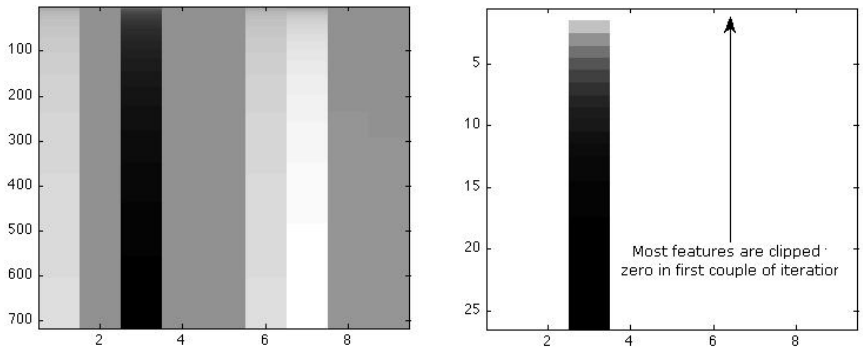

Fig. 3. Weights of lexicon entries of the words: *john*, *take*, *a*, *plane* and some noise.

The ℓ_1 training introduced some problems however. As we are trying to remove all irrelevant features, lexicon entries whose weights are equal to zero are removed. Fig. 3 shows that only one of the features is non-zero. This is because weights of correct lexicon entries that do not contribute to determining the probability of semantic representation being correct do not increase, but clip to zero instead. For the sentence "John takes a plane", due to the introduction of noise with regards to the semantics of the word *john*, only the correct semantic representation of the word contributes to determining whether a parse is correct or not.

7 Conclusion and Future Work

In this paper we presented a method of semantic parsing, which mapped natural language sentences to their semantic representations. In the lexical learning part the approach used an existing syntactic parser, inverse λ operators and generalization technique to learn semantics unknown words from syntax tree. In the parameter estimation part we reformulated the objective function to be regularized by ℓ_1-norm which introduced sparsity to the model and increased the convergence rate. We evaluated the approach on two benchmark corpus and the accuracy model showing that we outperformed many existing system in terms of accuracy while largely reduced the parameter estimation time and model size.

For future work, we expect even higher performance after introducing structured feature into our system. Simple ℓ_1-norm regularized models, however, do not take into consideration of complex structures within the features. A potential solution is to use group lasso [22] or grouped tree structure[23] learning. Such approaches would allow the domain-independent large-scale semantic parsing. We also want to extend the sentence-wise translation to discourse analysis, resulting in a higher level understanding of natural language.

References

1. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press (2003)
2. Kwiatkowski, T., Zettlemoyer, L., Goldwater, S., Steedman, M.: Inducing Probabilistic CCG Grammars from Logical Form with Higher-Order Unification. In Proc. of the 2010 Conf. on Empirical Methods in Natural Language Processing (EMNLP) (2010) 1223-1233
3. Zettlemoyer, L.S. , Collins, M. :Learning Context-dependent Mappings from Sentences to Logical Form. In Proc. of the Joint Conf. of the Association for Computational Linguistics and Int'l Joint Conf. on Natural Language Processing (ACL-IJCNLP) (2009)
4. Lu, W., Ng, H.T., Lee, W.S., Zettlemoyer, L.:A Generative Model for Parsing Natural Language to Meaning Representations. In Proc. of The Conf. on Empirical Methods in Natural Language Processing (EMNLP) (2008)
5. Zettlemoyer, L.S., Collins, M.: Online Learning of Relaxed CCG Grammars for Parsing to Logical Form. In Proc. of the Joint Conf. on Empirical Methods in

Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (2007)
6. Zettlemoyer, L.S., Collins, M.: Learning to Map Sentences to Logical Form: Structured Classification with Probabilistic Categorial Grammars. In Proc. of the 21st Conference on Uncertainty in Artificial Intelligence (UAI) (2005)
7. Ge, R., Mooney, R.J.:Learning a Compositional Semantic Parser using an Existing Syntactic Parser In the Joint Conf. of the 47th Annual Meeting of the Association for Comp. Ling. and the 4th Int'l Joint Conf. on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP) (2009) 611-619
8. Wong, Y.W., Mooney, R.J.: Learning Synchronous Grammars for Semantic Parsing with Lambda Calculus In Proc. of the 45th Annual Meeting of the Association for Computational Linguistics (ACL) (2007) 960-967
9. Wong, Y.W., Mooney, R.J.: Learning for Semantic Parsing with Statistical Machine Translation In Proc. of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT/NAACL) (2006) 439-446
10. Ge, R., Mooney. R.J.: A Statistical Semantic Parser that Integrates Syntax and Semantics. In Proc. of the Ninth Conference on Computational Natural Language Learning
11. Kate, R.J., Mooney, R.J.: Using String-Kernels for Learning Semantic Parsers In Proc. of the Joint 21st Int'l Conf. on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING/ACL) (2006) 913-920
12. Zelle, J.M., Mooney, R.J.: Learning to parse database queries using inductive logic programming. In Proc. of National Conference on Artificial Intelligence (AAAI) (1996)
13. Carpenter, B.: Typed-Logical Semantics. The MIT Press.
14. Steedman, M.: The Syntactic Process. The MIT Press.
15. Clark, S., Curran, J. R.: Log-linear models for wide-coverage CCG parsing. In Proc. of the SIGDAT Conf. on Empirical Methods in Natural Language Processing (2003)
16. Clark, S., Curran, J. R.: Wide-coverage efficient statistical parsing with CCG and log linear model. Computational Linguistics (2007)
17. Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge University Press (2004)
18. Bishop, C.M.: Pattern recognition and machine learning. Springer (2006)
19. Ng, A. Y.: Feature selection, ℓ_1 vs. ℓ_2 regularization and rotational invariance. In Proc. of the 21st Int'l Conf. on Machine Learning (ICML) (2004)
20. Carpenter, B.: Lazy sparse stochastic gradient descent for regularized multinomial logistic regression. Technical report, Alias-i (2008)
21. Tsuruoka, Y., Tsujii, J., Ananiadou, S.: Stochastic gradient descent training for L1-regularized log-linear models with cumulative penalty. In Proc. of the 4th Int'l Joint Conf. on Natural Language Processing of the AFNLP.(2009) 477-485
22. Liu, J., Ye, J.: Fast Overlapping Group Lasso. arXiv:1009.0306v1 (2010)
23. Liu, J., Ye, J.: Moreau-Yosida Regularization for Grouped Tree Structure Learning. In Proc. of Advances in Neural Information Processing Systems (NIPS) (2010)
24. [Hidden for review]: Using Inverse-λ and Generalization to Translate English to Formal Language. (Under Review)