Task-Specific Representation Learning for Web-scale Entity Disambiguation

Rijula Kar*, Susmija Reddy*,Sourangshu Bhattacharya*,
Anirban Dasgupta†, Soumen Chakrabarti‡
* IIT Kharagpur, India, †IIT Gandhinagar, India, ‡IIT Bombay, India
rijula.cse@iitkgp.ac.in, {jsreddy,sourangshu}@cse.iitkgp.ernet.in, anirbandg@iitgn.ac.in, soumen@cse.iitb.ac.in

Abstract

Named entity disambiguation (NED) is a central problem in information extraction. The goal is to link entities in a knowledge graph (KG) to their mention spans in unstructured text. Each distinct mention span (like John Smith, Jordan or Apache) represents a multi-class classification task. NED can therefore be modeled as a multitask problem with tens of millions of tasks for realistic KGs. We initiate an investigation into neural representations, network architectures, and training protocols for multitask NED. Specifically, we propose a task-sensitive representation learning framework that learns mention dependent representations, followed by a common classifier. Parameter learning in our framework can be decomposed into solving multiple smaller problems involving overlapping groups of tasks. We prove bounds for excess risk, which provide additional insight into the problem of multi-task representation learning. While remaining practical in terms of training memory and time requirements, our approach outperforms recent strong baselines, on four benchmark data sets.

Introduction and Related Work

Named entity disambiguation (NED) (Mihalcea and Csomai 2007; Cucerzan 2007; Milne and Witten 2008; Kulkarni et al. 2009; Hoffart and others 2011; Hoffart, Altun, and Weikum 2014; Lazic et al. 2015; Globerson et al. 2016; Li et al. 2016; Ganea and Hofmann 2017; Ji 2014) is a key problem in information extraction. The goal is to link canonical entity nodes (such as `en.wikipedia.org/wiki/Albert_Einstein`) in a knowledge graph (KG), which may have aliases like "Einstein" and "the dopey one", to their mention spans in unstructured text (such as "Albert"), using the mention context to disambiguate from other entities with similar aliases. For a given mention, NED is a multi-class classification problem, where the input is obtained from the mention context, and the output is an entity which has an alias matching the mention.

NED is a standard tool offered by IBM (IBM 2017), Google (Google 2017) and Microsoft (Microsoft 2017) cloud services. NED has seen sustained research since 2006, and much recent interest. (Durrett and Klein 2014) present a CRF model unifying NED with coreference resolution and entity type inference. (Lazic et al. 2015) describe Google's NED system, which uses a form of attention on context token features. (Globerson et al. 2016) describe an enhancement based on an attention mechanism on neighboring mentions. It outperforms the systems of (Durrett and Klein 2014) and (Lazic et al. 2015). Entity embeddings have proved useful to NED. (He et al. 2013) have used Wikipedia definition pages to embed entities. Like (Lazic et al. 2015), (Ganea and Hofmann 2017) have focused attention on specific context words while also embedding words and entities to get the best benchmark accuracy to date.

By regarding the disambiguation of each distinct mention string as a *task*, NED can be formulated as a multitask learning (MTL) problem. Ideally, a model for the mention *Jordan* that can distinguish people from rivers and countries should inform the model for *Washington*. Such information sharing between mentions is a very natural idea, but it is surprisingly unexplored. (Bunescu and Pasca 2006) demonstrated that a limited form of information sharing through types of candidate entities can improve accuracy. Ideally, MTL should be achieved without (being limited to) the explicit agency of types or other feature engineering. Of specific interest are MTL techniques that build shared representations by combining information from different tasks. Note that this form of sharing between tasks is quite distinct from *collective* NED (Kulkarni et al. 2009; Hoffart and others 2011; Ratinov et al. 2011; Globerson et al. 2016), which seeks to maximize a joint compatibility score over the entity labels chosen for the mentions in a document.

Recently, (Maurer, Pontil, and Romera-Paredes 2016) proposed and analyzed a multitask representation learning (MTRL) framework. They learned a common representation for instances of each task and used different classifiers for each task. Using this technique for NED requires large representation dimensions, which leads to massive parameter matrices. It also violates the intuition (Jin et al. 2014; Lazic et al. 2015) that each mention has only a few important features. Another baseline approach would be to randomly project (Weinberger et al. 2009) raw mention context features onto a lower dimensional space using random projections, followed by standard MTL techniques.

Our contributions: In this paper, we propose an unconventional MTL framework particularly suited to NED, called task-specific representation learning (TSRL). We first convert the raw input to a task-specific representation, and then apply a classifier shared within groups of tasks to get an entity label. Tasks can influence each other in flexible grouping schemes. In one such scheme, tasks (mentions) which share candidate labels (entities) can influence each other. Thus the problem naturally decomposes into overlapping groups of tasks with shared labels, thus helping manage extremely large task and label spaces with limited RAM. Our TSRL framework may be regarded as a generalization of (Maurer, Pontil, and Romera-Paredes 2016). Building on their analysis technique, we give bounds, in a multiclass setting, on the excess risk of our approach. Despite the significant reorganization of how network weights are shared, we show that the dependence on the number of tasks does not change from MTRL to TSRL. Also, the dependence on the number of labels is larger for the "classifier cost" (term containing complexity of final layer function class, $\mathcal{Q}$) as compared to the "representation cost" (term containing complexity of first layer function class, $\mathcal{G}(\mathcal{H})$).

We report extensively on the accuracy of TSRL for the NED task over the standard CoNLL, MSNBC, AQUAINT and ACE data sets (Hoffart and others 2011; Guo and Barbosa 2016). In terms of both micro- and macro-averaged accuracy, TSRL surpasses standard MTL and MTRL approaches, as well as the best feature-engineered baselines, in most cases. TSRL is particularly effective at boosting the accuracy of rare entities by 'borrowing' from easier tasks. Our code is available[1].

Methods

Problem definition and notation

In the NED task, we have to annotate **token spans** in unstructured text (e.g., "In May, Jordan scored more points than anyone else in NBA.") with a unique **entity** ID (here Jordan refers to the basketball player, not the Berkeley professor, country or river) from a knowledge graph (KG). The string Jordan is called the **mention**, which occurs in a longer textual **context** or mention **instance**. The context contains vital **features** (like 'scored' and 'NBA') which makes the NED task possible. The entity ID is the **label** to be predicted. Wikipedia will be our prototypical KG. It provides two kinds of data. First, it specifies a bipartite many-to-many relation between entities and their **aliases** (e.g., New York City is also called the Big Apple). An entity becomes a **candidate** label for a mention if the mention string closely matches at least one of its alias strings. Second, the KG gives us two forms of corpus text associated with each entity:

- Text in the entity's **definition page**, and
- Token spans linked to the entity, appearing on any page (which we call **gold mentions**).

Raw features extracted from these texts can be words in the context of a mention, the URL of the page where the mention occurs, topics (supervised or unsupervised) present in the page, other entities, etc. These features are collected into a raw mention context representation denoted x. The typical $x \in \mathcal{X} = \mathbb{R}^D$ is a sparse (hundreds of present features) vector in a space of millions of raw features (we call this number D).

[1]Visit `https://github.com/rijula/tsrl-aaai18` and `https://goo.gl/vw6C6g`

At first glance, disambiguating each distinct alias string (given context features) appears to be a multiclass classification problem. (For simplicity, here we will ignore the possibility that the true entity is outside the KG, i.e., the reject option. Various standard techniques are used to handle nil labels.) Suppose there are U distinct aliases in the KG, indexed $u \in [1, U]$. We can think of each distinct alias as a **task**, with its space of labels Y_u as the entities known by this alias. A NED system is given $\langle u, x\rangle$ as input and has to predict a $y \in Y_u$. Let $\mathcal{Y} = Y_1 \cup Y_2 \cdots \cup Y_U$ denote the global set of labels with $|\mathcal{Y}| = E$, the total number of KG entities. Note that entities can be shared across multiple tasks. Henceforth, we use *task* and *alias*, as well as *entity* and *label*, interchangeably.

In multi-task learning, we are interested in *jointly learning* models for performing multiple labeling tasks. Aliases u_1 = 'Jordan' and u_2 = 'Washington' can each refer to people, states, and lakes. Therefore, some form of representation sharing may facilitate one task to assist the other. Note that this form of interaction is quite distinct from **collective** NED. Collective NED algorithms seek to maximize some form of compatibility between the entity labels chosen throughout a document. E.g., if a document mentions both "Edward Lee" and "Michael Jordan", they are more likely to be professors at U.C. Berkeley, rather than a horror novelist and an American basketball player.

While multitask learning is a tempting approach to NED, NED poses several challenges specific to the application.

- The number of tasks is much larger than standard multi-task learning problems. U ranges into millions for Wikipedia alone, and can be orders of magnitude larger for commercial knowledge graphs.
- The total number of entities E may range between hundreds of thousands to millions, which makes it an extreme classification problem. However, depending on the task u, only a subset of the labels Y_u are admissible. The typical $|Y_u|$ ranges from a handful to hundreds. In public data sets involving Wikipedia, the occurrence-weighted average $|Y_u|$ is over 70 (i.e., popular aliases tend to be more ambiguous).
- Machine learning has been applied intensively to NED. Most recent systems (Ganea and Hofmann 2017; Lazic et al. 2015; Globerson et al. 2016; Ganea and Hofmann 2017) convey the intuition that a *small number of mention-dependent features* yields the best performance. However, this idea is contrary to popular MTRL settings (Maurer, Pontil, and Romera-Paredes 2016), where learning a single feature representation for multiple tasks is proposed and analyzed.

In the rest of this section, we will develop a series of formulations that introduce synergy among tasks in the NED setting, starting with none.

Per-alias classifier (PAC) baseline

It is possible to train a multi-class classifier (such as logistic regression or linear SVM) for every task, to use the raw (lexicalized) features in x directly. Let $\bar{Z} = Z_1 \cup \cdots \cup Z_U$ be the entire training dataset for named entity recognition, with Z_u

being the subset for u^{th} task. Here $Z_u = \{Z_{u1}, \ldots, Z_{un_u}\}$ and $Z_{ui} = (X_{ui}, Y_{ui})$, $Y_{ui} \in Y_u$, is the i^{th} example for the u^{th} task. For training, we optimize for each task u:

$$W_u^* = \arg\min_{W_u} \sum_{i=1}^{n_u} l(W_u; Z_{ui}) + \lambda \|W_u\|_F^2 \qquad (1)$$

Here l is the loss function, for which we use the multi-class hinge loss (Krammer and Singer 2001) W_u is a $D \times |Y_u|$ matrix with the j^{th} column as the weights for j^{th} class. This is one of our baseline methods (called per-alias classifier), and works very well in practice (see results). However, this approach does not benefit from related tasks on entities. Also, for task u, this results in $D\,|Y_u|$ model weights, or $D \sum_{u=1}^{U} |Y_u| = U\,\bar{E}\,D$ parameters overall, where $\bar{E} = \sum_u |Y_u|/U$ denotes the average number of possible labels per task. This exceeds RAM on even lavishly appointed servers. No plausible caching scheme for per-alias models exists. Therefore, such models are usually distributed over multiple servers. On the positive side, each of the tasks can be trained independently.

Multitask learning (MTL) with signhash

Signhash (Weinberger et al. 2009) is one of the earliest systems to apply feature hashing for large-scale multitask learning. While the original signhash multi-task learning was proposed for binary classification, here we describe a natural multi-class generalization for the same. Multi-task signhash introduces dependencies between tasks by assuming that the classifier parameter w can be expressed as the sum of a general parameter w_0 and the task specific parameters w_u. This in turn is shown to be equivalent to projecting each feature vector X_{ui} using two hashing functions $\phi_0(X_{ui})$ and $\phi_u(X_{ui})$, or using a single large model vector w_h. We use the second interpretation. Hence, for every feature vector X_{ui}, we compute the task specific hashing function $h : \mathbb{R}^D \to \mathbb{R}^R$ as $h_u(X_{ui}) = \phi_0(X_{ui}) + \phi_u(X_{ui})$. Hence, our multitask learning with signhash (MTL-SH) can be trained as $W^* =$

$$\text{argmin}_W \sum_{u=1}^{U} \sum_{i=1}^{n_u} l_u(W; (h_u(X_{ui}), Y_{ui})) + \lambda \|W\|_F^2.$$

Here, R is the representation dimension which is a crucial quantity for tradeoff between expressiveness of the model and scalability of the algorithm. W is the $R \times E$ parameter matrix with one parameter vector for each entity. The classification score for a given test point x from task u, and a test entity $y \in Y_u$ is computed as: $f(h_u(x), y) = W_y^T h_u(x)$. The final classification label is computed as: $y^* = \arg\max_{y \in Y_u} f(h_u(x), y)$. The loss function l_u is the multi-class hinge-loss with the class labels restricted to Y_u. Note that for a given task, parameters corresponding to non-admissible will neither be evaluated nor updated. We also note that the final score function for a given mention and entity pair $g_u(x, y)$ is a composition of two functions $g_u = f \circ h_u$, where h_u is a fixed random hash function.

Multi-task representation learning (MTRL)

The main drawback of the method described in the previous section is that the representation function h_u is a data-oblivious random function, and hence cannot be tuned to the dataset at hand. This problem is handled by representation learning in general (Bengio, Courville, and Vincent 2013) and multitask representation learning (MTRL) (Maurer, Pontil, and Romera-Paredes 2016), for the multitask learning setting, in particular. The predictor in (Maurer, Pontil, and Romera-Paredes 2016) outputs a single real number $g_u(x)$. Here we describe the multiclass generalization of the same.

Given a feature vector corresponding to a mention instance $x \in \mathbb{R}^D$ for a given task u, the scoring function $g_u : \mathbb{R}^D \to \mathbb{R}^{|\mathcal{Y}_u|}$, is defined as a composition of the representation function $h : \mathbb{R}^D \to \mathbb{R}^R$ and task specific scoring function $f_u : \mathbb{R}^R \to \mathbb{R}^{|\mathcal{Y}_u|}$: $g_u = f_u \circ h$. In our case, the functions h and f_u are implemented using layers of neural networks with ReLU activation functions and L_2-norm regularization. Hence, $h \in \mathcal{H}$ and $f_u \in \mathcal{F}$ where:

$$\mathcal{H} = \{h : h(x) = [W_h x]_+,\ W_h \in \mathbb{R}^{D \times R},\ |W_h|_F \leq \gamma_1\}$$
$$\mathcal{F} = \{f : f(x) = [W_f x]_+,\ W_f \in \mathbb{R}^{R \times |\mathcal{Y}|},\ |W_f|_F \leq \gamma_2\} \qquad (2)$$

Here $[.]_+$ is the component-wise maximum of the vector with 0. Note that, for uniformity, $\mathcal{F}$ produces a vector in $\mathbb{R}^{|\mathcal{Y}|}$, although only $|\mathcal{Y}_u|$ components are important for the u^{th} task. γ_1 and γ_2 are user defined parameters. The final label is predicted as $y^* = \arg\max_{y \in \mathcal{Y}_u} (g_u(x))_y$. Here, we are using the label y to index the corresponding element of vector $g_u(x)$. Hence, given the training dataset $\bar{Z}$ (described in the previous section), the optimal parameters are obtained by minimizing the overall loss function: $(h^*, f_1^*, \ldots, f_U^*) =$

$$\underset{h \in \mathcal{H}, (f_1, \ldots, f_U) \in \mathcal{F}^U}{\text{argmin}} \frac{1}{U} \sum_{u=1}^{U} \frac{1}{n_u} \sum_{i=1}^{n_u} l(f_u(h(X_{ui})), Y_{ui}) \qquad (3)$$

Note that for this model the number of parameters is $DR + R\sum_{u=1}^{U} |\mathcal{Y}_u| = (DR + RU\bar{E})$.

Task-specific Representation Learning (TSRL)

The main problem with the method described in the previous section is that the representation function h does not depend on task at hand. This has advantages such as invariance, coherence, sparsity, unsupervised learning, etc. (Bengio, Courville, and Vincent 2013). However, it lacks *parsimony* in terms of representation dimensions needed for classifying all tasks. For example, in order to classify the mention phrase `bank` as either river bank or financial bank, features such as `breeze, view, bridge, loan, account, transaction` etc might be important, whereas to differentiate `apple` (company) from `apple` (fruit), features like `seed, sweet, sour, mac, aesthetic, stock price`, etc might be useful. A single representation function capturing all such aspects for all mentions is likely to be high-dimensional, which will result in huge parameter space, leading to efficiency and optimization issues. Moreover, intuition from recent work (Ganea and Hofmann 2017; Lazic et al. 2015; Globerson et al. 2016) suggests that not only efficiency, but generalization performance of classifiers benefit from small number of task specific features. Hence we propose to make the representation function h_u depend on the task u.

However, in the MTRL setting described above, the information sharing across tasks happens using the shared representation function, since the scoring functions f_u are task-dependent. In our setting, using a task-dependent scoring function will result in the optimization problem decomposing over the tasks, hence forbidding information sharing across tasks. Therefore, we use a common classification function f across all tasks (or clusters of related tasks). Thus, our scoring function for task u, $g_u = f \circ h_u : \mathbb{R}^D \rightarrow \mathbb{R}^E$, and the entity label is calculated as: $y^* = \arg\max_{y \in \mathcal{Y}_u} (g_u(x))_y$. Given a training dataset $\bar{Z}$, the optimal parameters can be learnt by solving:

$$(h_1^*, \ldots, h_U^*, f^*) = \underset{(h_1,\ldots,h_U) \in \mathcal{H}^U, f \in \mathcal{F}}{\text{argmin}} \frac{1}{U} \sum_{u=1}^{U} \frac{1}{n_u} \sum_{i=1}^{n_u} l_u(f(h_u(X_{ui})), Y_{ui}) \quad (4)$$

Contrast the above form and model space against (3). We call this formulation the *task-specific representation learning* (TSRL) formulation.

We use a standard multiclass hinge loss $l_u(g_u(x), y) = \max(0, 1 + \max_{y' \in \mathcal{Y}_u} ((g_u(x))_{y'} - (g_u(x))_y))$, with the set of labels restricted according to task u. This requires $O(DUR + RE)$ parameters. Writing (4) in terms of model weights and adding regularization terms, we get:

$$(W_{h1}^*, \ldots, W_{hU}^*, W_f^*) = \underset{W_{h1},\ldots,W_{hU},W_f}{\text{argmin}} \left[\frac{1}{U} \sum_{u=1}^{U} \frac{1}{n_u} \sum_{i=1}^{n_u} l_u([W_f[W_{hu}X_{ui}]_+]_+, Y_{ui}) + \lambda_f \|W_f\|_F^2 + \lambda_h \sum_{u=1}^{U} \|W_{hu}\|_F^2 . \right] \quad (5)$$

λ_h and λ_f are related to γ_1 and γ_2 (eqn. 2) via duality. This problem can solved using standard optimization techniques, though the number of parameters can be prohibitively high for standard NED tasks. In the next section, we suggest a technique for dealing with this problem.

Task Grouping

Formulation (5) is stated in general terms, summing the loss over all u. Practical considerations limit what u's we may allow to influence each other. Even the modest-sized AIDA-CoNLL dataset (Hoffart and others 2011) has over 6000 mentions. Collecting all relevant training data from Wikipedia results in over a million instances across ∼25000 entity labels. The total number of parameters we can update in RAM at a time may limit us to smaller task groups. We may not even anticipate any accuracy benefit by bringing together tasks that do not interact with each other through overlapping entities, mentions, or context features.

A common characteristic in NED data is that a task u shares an entity with a relatively small number of tasks u'. This suggests the following technique for training (5):

1. Choose a primary task, say u.
2. Collect a group of *related* tasks C_u which have overlap of admissible entities with u.
3. Optimize (5) for only the subset of tasks C_u, and use the resulting model for task u.

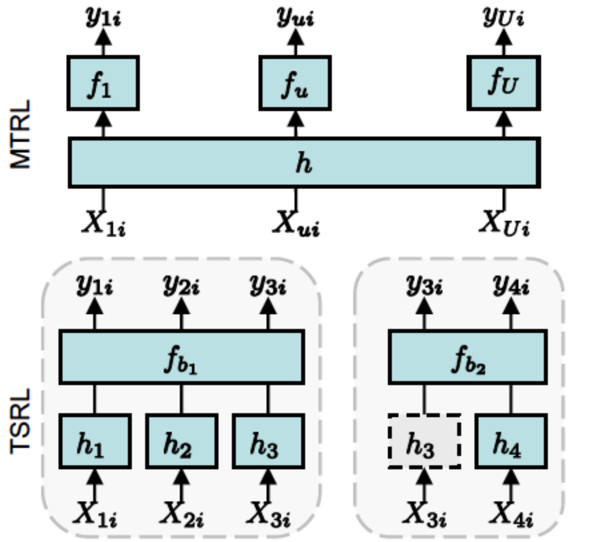

Figure 1: Difference between MTRL and TSRL. Mappings f, f_u, h, h_u are implemented by trained weights $W_\bullet$ as in the text. Task $u = 3$ is primary for group b_1 but not for b_2.

Two TSRL task groups b_1, b_2 are shown in Figure 1, which also contrasts the structures of MTRL and TSRL. $u = 3$ is primary for group b_1 but not b_2, although instances $\{(X_{3i}, y_{3i}\}$ can transfer information to $u = 4$. I.e., the dotted h_3 is not used for test instances of task 3. Within each group, the f-layer is capable of choosing, via softmax, any entity label in the union of all tasks in the group, but we limit prediction and loss computation to only the entities admissible to each task.

In view of the above mentioned task-grouping strategy, Equation (5) can be decomposed into groups as follows:

$$(W_{h1}(b)^*, \ldots, W_{hU}(b)^*, W_f(b)^*) = \underset{W_{h1},\ldots,W_{hU},W_f}{\text{argmin}} \left[\frac{1}{|\mathcal{U}_b|} \sum_{u \in \mathcal{U}_b} \frac{1}{n_u} \sum_{i=1}^{n_u} l_u([W_f[W_{hu}X_{ui}]_+]_+, Y_{ui}) + \lambda_f \|W_f\|_F^2 + \lambda_h \sum_{u=1}^{U} \|W_{hu}\|_F^2 . \right] \quad (6)$$

where $\mathcal{U}_b$ is the set of tasks in group b. Note that every group b may not have useful parameters $W_{hu}(b)^*$ for every task u, since the task may not be part of the group. The claim below formally states the claim for correctness of the task grouping strategy.

Claim 1. *For any task-group b with primary task $u \in \mathcal{U}_b$ and any entity y which is a candidate entity for u, the parameters $W_{hu}(b)^*$ and $[W_f(b)^*]_y$ (obtained using* (6)*), are respectively equal to W_{hu}^* and $[W_f^*]_y$, obtained using* (5)*; notwithstanding the non-uniqueness of solutions due to differences in starting points and example order.*

Proof. Our argument depends on the stochastic gradient descent algorithm, using per-example update scheme. Training instances that can affect W_{hu} (during an update), given the same $[W_f]_y$ for all candidate entities of task u, are present in all task groups which have task u. Also, examples which can affect $[W_f]_y$ for all candidate entities y of the primary task u, are present in the task-group by construction (since all tasks having candidate entities common to the primary task are added to the task group). Hence, for a given task-group b with primary task u and a candidate entity y of u, all examples which can affect $W_{hu}(b)^*$ and $[W_f(b)^*]_y$ are within the group b, thus showing their equivalence with global parameters. □

We have explored only one practical grouping scheme. Accuracy may be further enhanced by finding task groups using entities connected through types or relations in the knowledge graph, or exploiting corpus statistics. Such enhanced clustering is left for future work.

Analysis

In this section, we give a bound on the excess risk for our learning algorithm. The analysis is oblivious to the manner in which task grouping is done. The proof mostly follows the recent framework of establishing bounds on the multitask representation learning by (Maurer, Pontil, and Romera-Paredes 2016). Here, we focus on the changes that arise due to our modified architecture, as well as the multiclass setting. For simplicity, we assume that all tasks have the same number of target entities, as well as the same number of training examples, i.e. $\mathcal{Y}_u = E$ and $n_u = n$ for all u. Recall that the scoring function for task u is $g_u : \mathbb{R}^D \to \mathbb{R}^E$. The last stage of the architecture consists of the function class $\mathcal{F} = \mathcal{W}^E$ where each $\mathcal{W} \in \mathbb{R}^R \to \mathbb{R}$. We denote by $f \in \mathcal{F}$ a general function belonging to this class, and by $f(\cdot)_e \in \mathcal{W}$ the e^{th} component of f, for $e \in E$. Let L be the upper bound on the Lipschitz coefficient of $f(\cdot)_e$ for all e i.e.

$$L = \sup_{e \in E} \sup_{y \neq y' \in \mathbb{R}^R} \frac{f(y)_e - f(y')_e}{\|y - y'\|}$$

The following easy claim bounds the Lipschitz coefficient of $f(\cdot)$ in terms of L.

Lemma 2. *The Lipschitz coefficient of $f(\cdot) : \mathbb{R}^R \to \mathbb{R}^E$ is bounded by $L\sqrt{E}$.*

(Proof is described in appendix due to space constraint.) Given a set $Y \subset \mathbb{R}^D$, the *Gaussian average* of Y is defined as $G(Y) = \mathbb{E}_{\gamma \in \mathbb{R}^D}[\sup_{y \in Y} \langle \gamma, y \rangle]$ where each $\gamma \sim N(0,1)$ independently. Following the notation by (Maurer, Pontil, and Romera-Paredes 2016) we will often write $\mathbb{R}^{m \times n}$ as $\mathbb{R}^{mn}$ for ease of notation. We will use the following result by (Ando and Zhang 2005) that shows that the task averaged estimation error is not much smaller than its expectation.

Theorem 3. *Let Γ be a class of functions $\mathcal{X} \to [0,1]^U$ with $\gamma_u(\cdot)$ denoting the u^{th} component. Let $\{\mu_u, 1 \leq u \leq U\}$ be probability distributions on $\mathcal{X}$. Let $X_u \sim \mu_u^n$ be the dataset chosen for the u^{th} task, for all $u \in [1, U]$, where each $X_u = \{X_{ui}\}$ is a set of n examples. Let $\delta \in (0,1)$. Then, with probability $1 - \delta$ over the choice of $\cup_u X_u$,*

$$\frac{1}{U} \sum_u \mathbb{E}_{X \sim \mu_u} \gamma_u(X) - \frac{1}{Un} \sum_{u,i} \gamma_u(X_{ui}) \leq \frac{\sqrt{2\pi} G(S)}{nU} + \sqrt{\frac{9 \ln(2/\delta)}{nU}} \quad (7)$$

where $S \subset \mathbb{R}^{nU}$ is the set $\{\gamma_u(X_{ui}),\ \gamma \in \Gamma\}$.

Our theoretical measure of performance is the task averaged risk $\mathcal{E}_{\mathsf{avg}}(f, h_1, h_2, \ldots, h_U) =$

$$\frac{1}{U} \sum_{u=1}^{U} \mathbb{E}_{(X,Y) \sim \mu_u} \ell(f \circ h_u(X), Y)$$

The minimal expected risk $\mathcal{E}^*_{\mathsf{avg}}$ is thus the minimum over $(f, h_1, \ldots, h_U)$ of the above expression: $\mathcal{E}^*_{\mathsf{avg}} = \min_{f, h_1, \ldots, h_U} \mathcal{E}_{\mathsf{avg}}(f, h_1, h_2, \ldots, h_U)$. Define $Q(\mathcal{W})$ to be an upper bound on the Gaussian average of the Lipschitz coefficients of any of the components of $w(\cdot) \in \mathcal{W}$, i.e., $Q \equiv Q(\mathcal{W}) = \sup_{y \neq y', y, y' \in \mathbb{R}^{Rn}} \mathbb{E}_\gamma \left[\sup_{w \in \mathcal{W}} \frac{\sum_{i=1}^n \gamma_i (w(y_i) - w(y_i'))}{\|y - y'\|} \right]$. The following theorem will serve as the main step in our claim. The proof is in the appendix due to space constraints.

Theorem 4. *Assume $0 \in \mathcal{H}$ and $f(0) = 0$ for all $f \in \mathcal{F}$. Let $Z_u = \{Z_{ui} = (X_{ui}, Y_{ui})\}$ denote the examples drawn from $\{\mu_u\}$ for each $u \in [1, U]$. Let $\bar{Z} = \{Z_{ui}\}$ and $\bar{X} = \{X_{ui}\}$. Let $\delta \in (0,1)$. Then, with probability $1 - \delta$ over the draw of the samples $\cup_u Z_u$, there exist universal constants c_1 and c_2, such that that for every $f \in \mathcal{F}$ and every $(h_1, \ldots h_U) \in \mathcal{H}^U$, it holds that*

$$\mathcal{E}_{\mathsf{avg}}(f, h_1, h_2, \ldots, h_U) - \frac{1}{Un} \sum_{u,i} \ell(f \circ h_u(X_{ui}), Y_{ui}) \leq$$

$$\frac{c_1 L \sqrt{E} G(\mathcal{H}(\bar{X}))}{nU} + \frac{c_2 E Q \sup_{h \in \mathcal{H}} \|h(\bar{X})\|}{n\sqrt{U}} + \sqrt{\frac{9 \ln(2/\delta)}{nU}}$$

Our final bound on the excess risk is given by the following theorem. The proof of this follows directly from the proof of the corresponding theorem in (Maurer, Pontil, and Romera-Paredes 2016) and hence is omitted.

Theorem 5. *Let $Z_u, \bar{Z}, \bar{X}$ be defined as in Theorem 4 Let $f^*, \{h_u^*, u \in [1, U]\}$ denote the optimal solutions found by solving the optimization problem (4) over the training examples $\bar{Z}$. Let $\mathcal{H}, \mathcal{F}$ be defined as above. Then, for $\delta > 0$, with probability $1 - \delta$ over the draw of the training examples $\bar{Z}$, we have that*

$$\mathcal{E}_{\mathsf{avg}}(\hat{f}, \hat{h}_1, \ldots, \hat{h}_U) - \mathcal{E}^*_{\mathsf{avg}} \leq c_1 \frac{L\sqrt{E} G(\mathcal{H}(\bar{X}))}{nU} + c_2 \frac{QE \sup_{h \in \mathcal{H}} \|h(\bar{X})\|}{n\sqrt{U}} + \sqrt{\frac{9 \ln(4/\delta)}{nU}}$$

The above result is analogous to that of (Maurer, Pontil, and Romera-Paredes 2016). In the specific case when the functions $h_u(\cdot)$ are of the form used in this work, it is possible to show that our upper bound is of the same order as (Maurer, Pontil, and Romera-Paredes 2016). There the

Table 1: AIDA-CoNLL testb characteristics.

Unique mentions (tasks)	1685
Testing instances	4485
Training instances	10M
Training instances / mention	6K
Number of task groups	314
Max mentions in group	11
Min mentions in group	2
Training instances / group	34034
Number of raw features	16M
Number of entities	69959
Avg entities / mention	41

Table 2: Accuracy on CoNLL testb dataset.

Method	Micro-avg Accuracy	Macro-avg Accuracy
(Hoffart and others 2011)	82.29	82.02
(He et al. 2013)	84.82	83.37
(Lazic et al. 2015)	86.4	-
(Globerson et al. 2016)	87.9	-
(Ganea and Hofmann 2017)	88.8	-
PAC, gold mentions	85.9	89.1
MTL-SH, gold mentions	86.9	90.0
MTRL, gold mentions	82.0	86.7
TSRL, gold mentions	88.0	90.2
TSRL, gold+definitions	**89.4**	**91.9**

Methods	MSNBC	AQUAINT	ACE
(Fang et al. 2016)	81.2	88.8	85.3
(Ganea et al. 2016)	91	89.2	88.7
(Milne and Witten 2008)	78	85	81
(Hoffart and others 2011)	79	56	80
(Ratinov et al. 2011)	75	83	82
(Cheng and Roth 2013)	90	**90**	86
(Guo and Barbosa 2016)	92	87	88
(Ganea and Hofmann 2017)	93.7	88.5	88.5
TSRL	**94.2**	88.95	**89.06**

Table 3: Micro F1 score on AQUAINT, ACE, MSNBC.

first term on the RHS was interpreted as the "cost of learning the representation" whereas the second term was the "cost of learning the classifier". For our setting, the dependence of both terms on the number of tasks U is the same as the bound for original MTRL; it does not depend on which layer is task dependent. Our bounds also depend on E, the number of labels per task, and, as expected, grows with it. Notably, the increase in the cost of learning the representation is slower ($\sqrt{E}$) compared to the increase in the cost of learning the classifier (E).

Experiments and Results

Dataset preparation

We broadly followed the protocols of (Guo and Barbosa 2016)[2] (ACE, AQUAINT, MSNBC data) and (Hoffart and others 2011) (CoNLL 2003 data, testb test fold). We used the alias-entity mapping indexes created by (Ganea and Hofmann 2017)[3]. The training corpus was collected from the November 2016 Wikipedia dump[4]. Similar to (Lazic et al. 2015), we extracted all noun phrases in Wikipedia pages as mention context features. Table 1 gives details of training dataset.

Implementation details

All experiments were implemented in Theano 0.8.2 and run on a few Xeon servers with 32 cores and 96 GB RAM each. For optimization, we used SGD with minibatches of 1000 mention instances and learning rate $1/\sqrt{k}$ where k is the epoch number. Label predictions were made after averaging the model weights over the last 30 iterations, to remove noise. L2 regularizers were logarithmically grid-searched between 10^{-6} and 10^{6}, and reporting best accuracy achieved in the test dataset. Accuracy, micro and macro-averaged F1 are as defined in (Ganea and Hofmann 2017).

Using our task grouping technique described earlier, we got 314, 94, 279, 255 overlapping task groups in case of CoNLL, ACE, AQUAINT, MSNBC datasets. Each group of related mentions is trained separately from other groups, not only for TSRL, but also for MTRL and MTL-SH, since the parameters for all the mentions in all groups taken togather would not fit in RAM. In our implementations of MTL-SH, MTRL, TSRL, we used top 30 prior-sorted candidate entities as in (He et al. 2013; Ganea and Hofmann 2017) for each mention within a group. We did not use any candidate selection criteria for the per-alias classifier (PAC). While training one group we train only one mention per minibatch with only considering top 30 candidate entities.

Comparison with existing NED systems

We compare TSRL with some recent competitive NED baselines, as well as per-alias classifier (PAC), multitask learning using signhash (MTL-SH) (Weinberger et al. 2009) and multitask representation learning (MTRL) (Maurer, Pontil, and Romera-Paredes 2016). To avoid confounding inter-task signals with collective inference across a document or discourse (Ratinov et al. 2011; Durrett and Klein 2014; Globerson et al. 2016; Ganea and Hofmann 2017), we compare only local per-mention models. While collective NED generally yields better accuracies, our study clearly identifies the benefits of task sharing. TSRL can be readily embedded in collective models. Another detail is that local models can use gold mention contexts of entities, their definition page text from Wikipedia, or both. We have separated TSRL into gold-mention-only and gold-mention plus definition page text.

In Table 2, we report the accuracy by taking representation dimension (R) for MTL-SH as 1M and for MTRL and TSRL D and R is 1M and 100 respectively. MTL-SH wins over per-alias classifier (PAC), showing that there is inter-task information that can be exploited. The apparent anomaly of MTRL's poor performance is explained by the need to group tasks before training it, because of MTRL's extreme RAM requirements if all tasks are to be trained together, which is not possible at our scale. TSRL with only gold mention context features beats all baselines except (Ganea and Hofmann 2017). Adding entity definition page text features (as all other systems already do) makes TSRL the best local algorithm.

Table 3 compares TSRL against several recent strong baseline for the other three data sets, with similar trends: TSRL is the best for two and third best for the third data set.

[2]`https://bit.ly/2gnSBLg`

[3]`https://github.com/dalab/deep-ed`

[4]`https://dumps.wikimedia.org/enwiki/`

Table 4: CoNLL accuracy of PAC, MTL-SH, MTRL and TSRL while standardizing the number of model parameters.

		PAC	MTL-SH			MTRL			TSRL		
Cl No.	**No of Param**	**Acc**	R	E	**Acc**	R	**U**$\bar{E}$	**Acc**	R	E	**Acc**
C1	250005150	90.4	2427234	103	85.7	380	130	84.1	50	103	**90.4**
	500010300	90.4	4854468	103	85.7	630	130	84.1	100	103	**98.4**
C2	50006150	66.0	3252082	123	58.9	541	141	57.4	50	123	**71.1**
	100012300	66.0	6504165	123	58.9	941	141	57.4	100	123	**71.1**
C3	200004000	47.2	2500050	80	**72.2**	292	92	61.1	50	80	61.1
	400008000	47.2	5000100	80	**72.2**	492	92	61.1	100	80	**72.2**

Table 5: CoNLL Accuracy with respect to prior wise sorted entity ranking.

Model	**Acc in Bin-1**	**Acc in Bin-2**	**Acc in Bin-3**	**Acc in Bin-4**	**Acc in Bin-5**
PAC, gold mentions	93.9 %	62.4 %	63.4%	64 %	35 %
MTL-SH, gold mentions	94.5 %	67.3%	69.2%	64%	–
TSRL, gold mentions	**95.5%**	**68.53%**	**73%**	64%	–

Table 6: Training performance comparison.

	MTL-SH			MTRL					TSRL				
Cl No. Training size	E	Time(s)	NPU(M)	R	U$\bar{E}$	Time(s)	NPU(M) W_h	NPU(K) W_f	R	E	Time(s)	NPU(M) W_h	NPU(K) W_f
C1 47317	53	27	85	50	60	31	141.3	72	50	53	**25**	141.1	67
	53	**27**	85	100	60	37	282.7	144	100	53	29	282.2	134
	53	**27**	85	500	60	69	1413	720	500	53	61	1411	672
C2 162117	103	136	362	50	130	139	445.8	245	50	136	**101**	445.7	228
	103	136	362	100	130	155	891.6	490	100	136	**120**	891.4	457
	103	**136**	362	500	130	263	4458	2450	500	136	236	4457	2288
C3 252510	80	166	267	50	92	178	603.9	378	50	80	**139**	603.5	352
	80	166	267	100	92	196	1207	756	100	80	**154**	1207	705
	80	**166**	267	500	92	326	6039	3781	500	80	274	6035	3529

Accuracy against number of model parameters

For the rest of the section we focus on the CoNLL data set. While end-to-end comparisons of TSRL against published prior art stand as-is, we need more care comparing TSRL against PAC, MTL-SH and MTRL, to make sure the amount of RAM occupied by various models are comparable. Let the input dimension of x_{ui} be D, representation dimension be R, and number of entities of the task-group be E and the total number of mention-entities is $U\bar{E}$. The the number of parameters to be learned by MTL is RE, MTRL is $DR + RU\bar{E}$ and TSRL is $DUR + RE$. It is obvious that $U\bar{E} > E$. (Here we assume parameter vectors are dense.) We chose three representative task-groups C1, C2, and C3. For each, we set R of TSRL to 50 and 100. Then we chose R for MTL-SH and MTRL so that the RAM requirements of the three methods are (almost) equal. D for MTRL and TSRL is 1 million for these task-groups.

Table 4 shows the results. Task-group C3 has relatively few candidates. This helps MTL-SH allocate relatively larger R and attain competitive accuracy (when R for TSRL is very small). However, on task-groups C1 and C2 with more candidates, TSRL wins over other multi-task techniques. TSRL always wins for $R = 100$.

Accuracy for entities binned by prior probability

Accuracy for an entity e is the percentage of mention-instances, for which the true entity label is e, labeled correctly. A potential benefit of TSRL is that rare entities can gain accuracy due to borrowing of instances from related mentions. To test this, we divided entities into five bins according to their mention priors. `Bin-1` contains the 1^{st} ranked entity (by mention prior). Similarly, `Bin-2`, `Bin-3`, `Bin-4`, and `Bin-5` contain entities ranked 2 – 10, 11 – 20, 21 – 30, and > 31 respectively. Also, the bins contain 3306, 1068, 52, 28, and 31 instances respectively. As described above, no entities were in Bin-5 while training of MTL-SH and TSRL. Table 5 compares accuracy of per-alias classification (PAC) against MTL-SH and TSRL disaggregated into bins. Apart from retaining accuracy at large priors, we see clear benefits for entities with lower priors.

Comparison of training speed

Training performance of different models are given by time (in secs) and number of parameter updates (NPUs). Time taken for training a task-group depends on the number of training instances in the group, and the number of hidden units in the model. On the other hand, NPUs depends on the sparsity of training instances in a task-group, the number of hidden units and the number of classes. Three representative task-groups are shown in Table 6. The dimension D is 1M for all. We fix the hidden units (R) of our model and MTRL and compare time and NPU against baselines, taking the average of the first 30 epochs. As MTL-SH does not learn any intermediate representation, the time and space requirements remains same for any hidden units of MTRL and TSRL. From the scalability point of view when the number of representation dimension(R) to be learned is less than number of classes, TSRL outperforms the baselines, given input dimension of all the baselines are the same.

Anecdotal evidence of inter-task transfer

Different runs for PAC, MTL, MTRL and TSRL produce network weights W_h, W_f that are incomparable to each other, leaving us with only end-to-end accuracy to compare. To collect strong circumstantial evidence of inter-task transfer, we identified mentions whose accuracy improved substantially from PAC to TSRL. For these, we checked if they had only a few training instances, but another mention placed in their group had many more. This is often true; samples are shown in Table 7.

Table 7: Circumstantial evidence of instance-poor tasks gaining accuracy by sharing W_f slices with instance-rich tasks.

Instance-rich Mention				Instance-poor Mention			
Mention	Training Instances	PAC Acc	TSRL Acc	Mention	Training Instances	PAC Acc	TSRL Acc
Brazil	95186	100	100	Brazilian	1180	0	100
Italy	105951	92	100	Italians	158	60	100
John Gorst	29	100	100	Gorst	22	0	100
Germany	165558	93.8	100	German	62670	53	93
Australia	137487	46	73.4	Australian	1438	69.2	100

Conclusion

We investigated MTL and MTRL for NED. Conventional MTRL first obtains a task-independent instance representation $h(x)$, then applies a task-specific predictor $f_u(h(x))$. In contrast, we propose, justify, analyze, and evaluate a task-sensitive representation $h_u(x)$ followed by a global predictor $f(h_u(x))$. Our framework allows a flexible grouping of tasks to fit in RAM. We show the superiority of TSRL to recent competitive baselines as well as direct application of standard MTL and MTRL. Our system boosts accuracy of rarer entities and mentions via inter-task sharing, resulting in larger gains for macroaveraged accuracy.

Acknowledgements

RK and SB acknowledge support of Govt. of India, DST, SERB grant number SB/S3/EECE/0249/2014. SC acknowledges an nVidia grant. AD acknowledges a Google faculty grant and a Cisco research grant.

References

Ando, R. K., and Zhang, T. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *JMLR* 6(Nov):1817–1853.

Bengio, Y.; Courville, A.; and Vincent, P. 2013. Representation learning: A review and new perspectives. *IEEE PAMI* 35(8):1798–1828.

Bunescu, R., and Pasca, M. 2006. Using encyclopedic knowledge for named entity disambiguation. In *EACL*, 9–16.

Cheng, X., and Roth, D. 2013. Relational inference for wikification. In *EMNLP Conference*, 1787–1796.

Cucerzan, S. 2007. Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP Conference*, 708–716.

Durrett, G., and Klein, D. 2014. A joint model for entity analysis: Coreference, typing, and linking. *TACL* 2:477–490.

Fang, W.; Zhang, J.; Wang, D.; Chen, Z.; and Li, M. 2016. Entity disambiguation by knowledge and text jointly embedding. In *CoNLL*, 260–269.

Ganea, O.-E., and Hofmann, T. 2017. Deep joint entity disambiguation with local neural attention. *arXiv preprint arXiv:1704.04920*.

Ganea, O.-E.; Ganea, M.; Lucchi, A.; Eickhoff, C.; and Hofmann, T. 2016. Probabilistic bag-of-hyperlinks model for entity linking. In *WWW Conference*, 927–938.

Globerson, A.; Lazic, N.; Chakrabarti, S.; Subramanya, A.; Ringgaard, M.; and Pereira, F. 2016. Collective entity resolution with multi-focal attention. In *ACL Conference*, 621–631.

Google. 2017. Cloud natural language api, google cloud platform. `https://cloud.google.com/natural-language/`.

Guo, Z., and Barbosa, D. 2016. Robust named entity disambiguation with random walks. *Semantic Web* 1–21.

He, Z.; Liu, S.; Li, M.; Zhou, M.; Zhang, L.; and Wang, H. 2013. Learning entity representation for entity disambiguation. In *ACL Conference*, 30–34.

Hoffart, J.; Altun, Y.; and Weikum, G. 2014. Discovering emerging entities with ambiguous names. In *WWW Conference*, 385–396. ACM.

Hoffart, J., et al. 2011. Robust disambiguation of named entities in text. In *EMNLP Conference*, 782–792. Edinburgh, Scotland, UK: SIGDAT.

IBM. 2017. Natural language understanding overview (IBM Watson developer cloud). `https://www.ibm.com/watson/developercloud/doc/natural-language-understanding/`.

Ji, H. 2014. Kbp 2016 entity discovery and linking. `http://nlp.cs.rpi.edu/kbp/2014/elreading.html`.

Jin, Y.; Kiciman, E.; Wang, K.; and Loynd, R. 2014. Entity linking at the tail: Sparse signals, unknown entities and phrase models. In *WSDM Conference*, 453–462.

Krammer, K., and Singer, Y. 2001. On the algorithmic implementation of multi-class SVMs. *Proc. of JMLR* 265–292.

Kulkarni, S.; Singh, A.; Ramakrishnan, G.; and Chakrabarti, S. 2009. Collective annotation of Wikipedia entities in Web text. In *SIGKDD Conference*, 457–466.

Lazic, N.; Subramanya, A.; Ringgaard, M.; and Pereira, F. 2015. Plato: A selective context model for entity resolution. *TACL* 3:503–515.

Li, Y.; Tan, S.; Sun, H.; Han, J.; Roth, D.; and Yan, X. 2016. Entity disambiguation with linkless knowledge bases. In *WWW Conference*, 1261–1270.

Maurer, A.; Pontil, M.; and Romera-Paredes, B. 2016. The benefit of multitask representation learning. *JMLR* 17(81):1–32.

Microsoft. 2017. Microsoft cognitive services: Entity linking intelligence service. `https://azure.microsoft.com/en-us/services/cognitive-services/entity-linking-intelligence-service/`.

Mihalcea, R., and Csomai, A. 2007. Wikify!: linking documents to encyclopedic knowledge. In *CIKM Conference*, 233–242. ACM.

Milne, D., and Witten, I. H. 2008. Learning to link with wikipedia. In *CIKM Conference*, 509–518. ACM.

Ratinov, L.; Roth, D.; Downey, D.; and Anderson, M. 2011. Local and global algorithms for disambiguation to Wikipedia. In *ACL Conference*, ACL/HLT, 1375–1384.

Weinberger, K.; Dasgupta, A.; Langford, J.; Smola, A.; and Attenberg, J. 2009. Feature hashing for large scale multitask learning. In *ICML*, 1113–1120.