USE OF KERNEL DEEP CONVEX NETWORKS AND END-TO-END LEARNING FOR SPOKEN LANGUAGE UNDERSTANDING

Li Deng, Gokhan Tur, Xiaodong He, and Dilek Hakkani-Tur

Microsoft Research

{deng|gokhant|xiaohe|dilekha}@microsoft.com

ABSTRACT

We present our recent and ongoing work on applying deep learning techniques to spoken language understanding (SLU) problems. The previously developed deep convex network (DCN) is extended to its kernel version (K-DCN) where the number of hidden units in each DCN layer approaches infinity using the kernel trick. We report experimental results demonstrating dramatic error reduction achieved by the K-DCN over both the Boosting-based baseline and the DCN on a domain classification task of SLU, especially when a highly correlated set of features extracted from search query click logs are used. Not only can DCN and K-DCN be used as a domain or intent classifier for SLU, they can also be used as local, discriminative feature extractors for the slot filling task of SLU. The interface of K-DCN to slot filling systems via the softmax function is presented. Finally, we outline an end-to-end learning strategy for training the softmax parameters (and potentially all DCN and K-DCN parameters) where the learning objective can take any performance measure (e.g. the F-measure) for the full SLU system.

Index Terms **—** kernel learning, deep learning, spoken language understanding, domain detection, slot filling

1. INTRODUCTION

In recent years, machine learning has been playing increasingly important roles in speech and language processing. In particular, deep learning techniques have significantly improved the state of the art on phone recognition (Deng et al, 2012; Mohamed et al, 2010, 2012), speech feature coding (Deng et al., 2010), and large vocabulary speech recognition (Dahl et al, 2012; Hinton et al, 2012; Monga et al., 2012).

Spoken language understanding (SLU) in human/machine spoken dialog systems aims to automatically identify the domain and intent of the user as expressed in natural language and to extract associated arguments or slots to achieve a goal. Deep learning was more recently demonstrated to be effective for spoken language understanding (SLU) by Tur et al (2012). In both areas of intent determination (or domain detection) and slot filling, the recent state of art has been based on discriminative classifiers such as Boosting or SVM (Tur and De Mori, 2011) and CRF (Raymond and Riccardi 2007; Hahn et al, 2011). The deep learning technique of deep convex network (DCN) developed in (Deng and Yu, 2011; Deng et al., 2012) was successfully applied to a domain-detection task, improving the state of the art approach using Boosting (Tur et al, 2012).

In this paper, we will present the kernel version of the DCN (which we call K-DCN), a significant extension of the previous DCN technique where the number of hidden units in each DCN layer approaches *infinity* using the kernel trick. We demonstrate the much better classification performance of the K-DCN over both the Boosting-based baseline and the DCN on a domain classification task.

In addition to domain or intent determination, another key task of SLU is slot-filling, which requires sequence modeling where a slot tag is assigned to each word or phrase in the input utterance. That is, the task is to find an optimal slot ID sequence. In this paper, we will discuss how to build sequential models for slot-filling using DCN and K-DCN to provide local features via a softmax-layer interface. We will also provide an end-to-end learning framework in which the features extracted discriminatively by DCN and K-DCN can be exploited to optimize a full SLU system that performs slot-filling tasks. This end-to-end training strategy allows us to optimize the slot-filling performance such as an F-measure directly.

2. FEATURE EXTRACTION FOR SLU

In this section, we review the DCN architecture and learning, and then extend it to the kernel version (resulting in K-DCN) by constructing infinite-dimensional hidden representations in each of the DCN module using the kernel trick. Both DCN and K-DCN can be used as powerful classifiers for domain detection and also as discriminative feature extractors for subsequent slot filling at the full-utterance level.

2.1 Deep convex network (DCN): A review

Here, we provide an overview of the DCN architecture. The philosophy of DCN's architecture design rests in the concept of stacking, where simple modules of functions or classifiers are composed first and then they are "stacked" on top of each other so as to learn complex functions or classifiers. Following this philosophy, Deng and Yu (2011) and Deng et al. (2012) developed and presented the basic DCN architecture that consists of many stacking modules, each of which takes a simplified form of shallow multilayer perceptron using convex optimization for learning perceptron weights. Fig. 1 gives an example of a three-block DCN, each consisting of three layers and each illustrated with a separate color. All hidden layers are sigmoid nonlinear. Prediction and input layers are linear. The DCN weight parameters $\boldsymbol{W}$ and $\boldsymbol{U}$ in each module are learned efficiently from training data. For making connections to the K-DCN in Section 2.2, we now review the

learning method for $\boldsymbol{U}$ given fixed $\boldsymbol{W}$ and hence fixed hidden units' outputs, which is $\boldsymbol{h}_i = \sigma(\boldsymbol{W}^T \boldsymbol{x}_i)$ at the bottom module of DCN and $\boldsymbol{h}_i^{(l)} = \sigma\left(\boldsymbol{W}^T [\boldsymbol{x}_i \,\middle|\, \boldsymbol{y}_i^{(l-1)}, \boldsymbol{y}_i^{(l-2)} \middle| .. \boldsymbol{y}_i^{(1)}]\right)$ at a higher module.

The learning objective of DCN is mean square error regularized by L_2 norm of the weights:

$$J(\boldsymbol{U}) = \tfrac{1}{2} Tr[(\boldsymbol{Y} - \boldsymbol{T})(\boldsymbol{Y} - \boldsymbol{T})^T] + \tfrac{C}{2} \boldsymbol{U}^T \boldsymbol{U} \quad (1)$$

where $\boldsymbol{y}_i = \boldsymbol{U}^T \boldsymbol{h}_i$ is DCN's output (for each module), $\boldsymbol{T} = [\boldsymbol{t}_1, \cdots, \boldsymbol{t}_i, \cdots, \boldsymbol{t}_N]$ are the target vectors for training, and C is the regularization parameter. The solution is well known:

$$\boldsymbol{U} = (C\boldsymbol{I} + \boldsymbol{H}\boldsymbol{H}^T)^{-1} \boldsymbol{H}\boldsymbol{T}^{\mathrm{T}}$$

Hence the output of DCN of each module can be written as

$$\boldsymbol{y}_i = \boldsymbol{T}\boldsymbol{H}^{\mathrm{T}} (C\boldsymbol{I} + \boldsymbol{H}\boldsymbol{H}^T)^{-1} \boldsymbol{h}_i \quad (2)$$

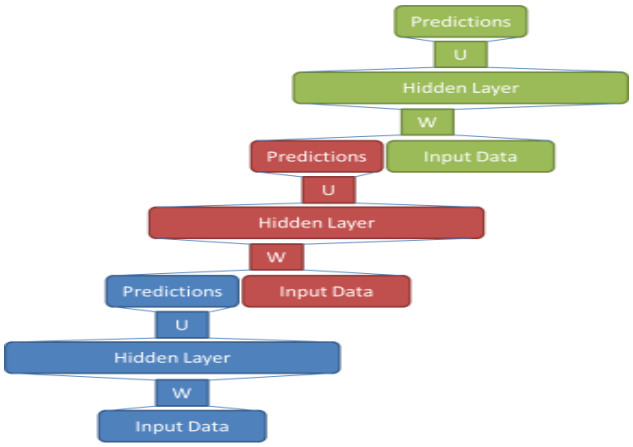

Figure 1: *A typical DCN architecture. Hidden layers are sigmoid nonlinear. Prediction and input layers are linear. Three modules are shown, each with a distinct color.*

2.2 Kernel deep convex network (K-DCN)

The DCN architecture reviewed above has convex learning for weight matrix U given the hidden layers' outputs in each module, but the learning of weight matrix $\boldsymbol{W}$ is non-convex. For most applications, the size of U is comparable to that of $\boldsymbol{W}$ and then DCN is not strictly a convex network. In a recent extension of DCN, a tensor structure was imposed, shifting the majority of non-convex learning burden for $\boldsymbol{W}$ into a convex one (Hutchinson et al, 2012). In the current K-DCN extension, we completely eliminate non-convex learning for $\boldsymbol{W}$ using the kernel trick (Hofmann et al, 2008).

To derive the K-DCN architecture and the associated learning algorithm, we first take the bottom module of DCN as an example and generalize the sigmoidal hidden layer $\boldsymbol{h}_i = \sigma(\boldsymbol{W}^T \boldsymbol{x}_i)$ in the DCN module into a generic nonlinear mapping function $\boldsymbol{G}(\boldsymbol{X})$ from the raw input feature $\boldsymbol{X}$, with high dimensionality in $\boldsymbol{G}(\boldsymbol{X})$ (possibly infinite) determined only implicitly by a kernel function to be chosen. Second, we reformulate the unconstrained optimization problem of (1) into a constrained one:

$$\begin{aligned} &\textit{minimize} \;\; \tfrac{1}{2} Tr[\boldsymbol{E}\boldsymbol{E}^{\mathrm{T}}] + \tfrac{C}{2} \boldsymbol{U}^{\mathrm{T}} \boldsymbol{U} \\ &\textit{subject to} \;\; \boldsymbol{T} - \boldsymbol{U}^T \boldsymbol{G}(\boldsymbol{X}) = \boldsymbol{E} \end{aligned}$$

Third, we make use of dual representations (rf. pages 293-294 in Bishop, 2006) of the above constrained optimization problem to obtain $\boldsymbol{U} = \boldsymbol{G}^{\mathrm{T}} \boldsymbol{a}$ where vector $\boldsymbol{a}$ takes the following form

$$\boldsymbol{a} = (C\boldsymbol{I} + \boldsymbol{K})^{-1} \boldsymbol{T}$$

and where $\boldsymbol{K} = \boldsymbol{G}(\boldsymbol{X})\boldsymbol{G}^{\mathrm{T}}(\boldsymbol{X})$ is a symmetric kernel matrix with elements of $K_{nm} = g^{\mathrm{T}}(x_n) g(x_m)$.

Finally, for each new input vector $\boldsymbol{x}$ in the test or dev set, we obtain the K-DCN (bottom) module's prediction of

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{U}^{\mathrm{T}} \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{a}^{\mathrm{T}} \boldsymbol{G}(\boldsymbol{X}) \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{k}^{\mathrm{T}}(\boldsymbol{x}) (C\boldsymbol{I} + \boldsymbol{K})^{-1} \boldsymbol{T} \quad (3)$$

where the kernel vector $\boldsymbol{k}(\boldsymbol{x})$ is so defined that its elements have values of $k_n(\boldsymbol{x}) = k(\boldsymbol{x}_n, \boldsymbol{x})$ in which $\boldsymbol{x}_n$ is a training sample and $\boldsymbol{x}$ is the current test sample.

For l-th module in K-DCN where $l \geq 2$, Eq. (3) holds except the kernel matrix is modified to

$$\boldsymbol{K} = \boldsymbol{G}([\boldsymbol{X} \,\middle|\, \boldsymbol{Y}^{(l-1)} \middle| \boldsymbol{Y}^{(l-2)} \middle| .. \boldsymbol{Y}^{(1)}]) \boldsymbol{G}^{\mathrm{T}}([\boldsymbol{X} \,\middle|\, \boldsymbol{Y}^{(l-1)} \middle| \boldsymbol{Y}^{(l-2)} \middle| .. \boldsymbol{Y}^{(1)}])$$

Comparing the prediction of (2) in DCN and of Eq. (3) in K-DCN, we see key advantages of K-DCN as follows. First, unlike DCN which need to compute hidden units' output H show in (2), K-DCN does not need to explicitly compute hidden units' output $\boldsymbol{G}(\boldsymbol{X})$ or $\boldsymbol{G}([\boldsymbol{X} \,\middle|\, \boldsymbol{Y}^{(l-1)} \middle| \boldsymbol{Y}^{(l-2)} \middle| .. \boldsymbol{Y}^{(1)}])$. In the experiments reported in Section IV, we have explored the use of Gaussian kernel, where kernel trick equivalently gives us an infinite number of hidden units without the need to compute them explicitly. Further, we no longer need to learn the lower-layer weight matrix $\boldsymbol{W}$ in DCN (Deng et al, 2012) and the kernel parameter (e.g., the single variance parameter σ in the Gaussian kernel) makes K-DCN much less subject to overfitting than DCN. In Fig. 2, we show the architecture of a K-DCN using the Gaussian kernel.

The entire K-DCN is characterized by two module-dependent hyper-parameters: $\sigma^{(l)}$ and $C^{(l)}$, the kernel smoothing parameter and regularization parameter. While both parameters are intuitive and their tuning (via line search or leave-one-out cross validation) is straightforward for a single bottom module, tuning them from module to module is more difficult. For example, if the bottom module is tuned too well, then adding more modules would not benefit much. In contrast, when the lower modules are loosely tuned (i.e., relaxed from the results obtained from straightforward methods), the overall K-DCN often performs much better. The experimental results reported in Section IV have been obtained using a set of empirically determined tuning schedules to adaptively regulate the K-DCN from bottom to top modules.

Without stacking to form a deep architecture, the use of kernel functions to perform nonlinear regression and classification has been reported in statistics and machine learning literature under a number of different names including kernel ridge regression, least-square SVM, kernel fisher discriminant, empirical kernel map, regularized least square classifier, extreme learning machine, and kernel partial least squares regression (e.g. Rosipal and Trejo, 2001; Huang et al, 2012; Karasuyama, M. and Nakano, 2008; Chen and Haykin, 2002; Kadri et al, 2011; Hofmann et al, 2008; Saunders et al, 1998). The key contribution of this work is to use this type of shallow machines as building blocks to construct a deep architecture. Importantly, we have discovered that the principles used to regularize a single shallow block are very different from those for the deep network consisting of many stacking blocks.

As a summary, the K-DCN described in this section has a set of highly desirable properties from the machine learning and pattern recognition perspectives. It combines the power of deep learning and kernel learning in a principled way and unlike the previous

DCN/DSN there is no non-convex optimization for K-DCN. The computation steps shown in Eq. (3) make K-DCN easier to scale up for parallel computing in distributed servers than the previous DCN and tensor-DSN. There are many fewer parameters in K-DCN to tune than DCN, T-DSN, and DNN, and there is no need for pre-training with often slow, empirical procedures related to RBM and DBN. Also, we have found that regularization plays a much more important role in K-DCN than in DCN and Tensor-DSN, and the effective regularization schedules developed sometimes can have intuitive insight and can be motivated by optimization tricks. Further, we have found empirically that K-DCN does not require data normalization, as is often essential in other deep networks such as DNN and DCN. Finally, our experience showed that, unlike other methods, K-DCN can easily handle mixed binary and continuous-valued inputs without data and output calibration. All these desirable properties have been demonstrated in our experiments on intent determination tasks to be described in Section 5.

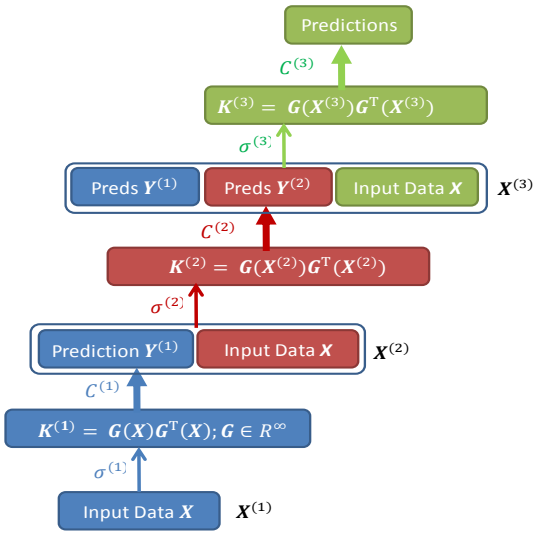

Figure 2: *Architecture illustration of K-DCN with three modules*

3. SPOKEN LANGUAGE UNDERSTANDING

Semantic parsing of input utterances typically consists of 3 tasks, domain detection, intent determination, and slot filling. Originated from call routing systems, domain detection or intent determination tasks are typically treated as semantic utterance classification, and originated from natural language to semantic template filling systems such as the DARPA ATIS, slot filling task is typically treated as sequence classification. Syntactic, semantic, and other contextual features are employed in statistical modeling of these SLU tasks (Tur and De Mori 2011).

An example sentence with domain, intent, and slot annotations, along with typical domain-independent named entities, is provided below, following the popular in/out/begin (IOB) representation, where *Boston* and *NewYork* are the departure and arrival cities specified as the slot values in the user's utterance, respectively.

	show	*flights*	*from*	*Boston*	*to*	*New*	*York*	*today*
Slots	O	O	O	B-dept	O	B-arr	I-arr	B-date
Names	O	O	O	B-city	O	B-city	I-city	O
Intent	*Find_Flight*							
Domain	*Airline Travel*							

4. SEMANTIC UTTERANCE CLASSIFICATION

The semantic utterance classification (SUC) task aims at classifying a given speech utterance X_r into one of M semantic classes, $\hat{C}_r \in C = \{C_1, \ldots, C_M\}$ (where r is the utterance index). Upon the observation of X_r, $\hat{C}_r$ is chosen so that the class-posterior probability given X_r, $P(C_r|X_r)$, is maximized. More formally,

$$\hat{C}_r = arg \max_{C_r} P(C_r|X_r)$$

Semantic classifiers need to allow significant utterance variations. A user may say *"I want to fly from San Francisco to New York next Sunday"* and another user may express the same information by saying *"Show me weekend flights between JFK and SFO"*. On the other hand, the command *"Show me the weekend snow forecast"* should be interpreted as an instance of another semantic domain class, say, "*Weather*." In order to do this, the selection of the feature functions $fi(C,W)$ aims at capturing the relation between the class C and word sequence W. Typically, binary or weighted n-gram features, with $n = 1, 2, 3$, to capture the likelihood of the n-grams, are generated to express the user intent for the semantic class C. Because of the very large dimensions of the input space, large margin classifiers such as SVMs or Boosting were found to be very good candidates for this task.

5. SEQUENCE MODELING FOR SLOT FILLING

In addition to domain detection and intent determination, another key task in SLU is slot-filling. Traditional sequential models for slot-filling include HMM, MEMM, SMT, and CRF (Hahn et al, 2011). Most of these models are based on discrete or discretized features. In this paper, we introduce a slot-filling model based on the log-linear framework, with dense continuously-valued features transformed from raw binary lexical features using DCN and K-DCN. We also describe how the model can be trained to directly maximize the accuracy metric for evaluation, where comprehensive experimental work is in progress.

5.1 Log-linear modeling for slot-filling

Given the observation, e.g., the input sentence, O, the optimal sequence of slot IDs $\hat{S}$ is obtained according to

$$\hat{S} = \underset{S}{\operatorname{argmax}} P(S|O) \quad (4)$$

where $P(S|O)$ is modeled by a log-linear model (similar to the approach by Macherey et al, 2009):

$$P(S|O) = \frac{1}{Z} exp\left\{\sum_{m=1}^{M} \lambda_m \log h_m(S,O)\right\} \quad (5)$$

and $Z = \sum_S exp\{\sum_m \lambda_m \log h_m(S,O)\}$ is the normalization denominator. M is the number of feature functions. Note that we define the feature functions $\{h_m(S,O)\}$ in log domain to simplify the notation in later sections.

In the log-linear model, the feature weights $\boldsymbol{\lambda} = \{\lambda_m\}$ are usually tuned by MERT on a held-out development set (Och 2003). In the following sub-sections, we will describe the actual feature models for slot-filling and the related learning problem.

5.2. Slot translation model

Assuming that the input sentence consists of K words, we design the word-to-slot translation feature which is scored as:

$$h_1(S,O) = \prod_k p(s_k|o_k) \tag{6}$$

where s_k and o_k are the k-th slot ID in sequence S and the k-th word in observation sentence O, respectively.

Instead of modeling translation probabilities directly, we take into account an n-gram context around the word o_k, and extract a local feature vector from that n-gram using the K-DCN.

Let us denote the local feature vector extracted by K-DCN by $\boldsymbol{x}$. Then we model the probability of slot ID i given feature $\boldsymbol{x}$ using the softmax function:

$$p(s=i|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}_i \boldsymbol{x}}}{\sum_i e^{\boldsymbol{w}_i \boldsymbol{x}}} \tag{7}$$

where $\boldsymbol{w}_i$ is the i-th row of the parameter matrix $\mathbf{W}$. Matrix $\mathbf{W}$ has a total of I rows and D columns, where I is the number of slot categories, and D is the dimension of the feature vector. In latter sections we describe how $\mathbf{W}$ can be learned in an end-to-end optimal manner.

5.3. Slot transition model

In order to capture the dependence between slot IDs, we also design additional "features" based on the slot-transition model:

$$h_2(S,O) = \prod_k p(s_k|s_{k-1}) \tag{8}$$

which serves as a bi-gram language model (LM) for the slot IDs. In our work, this bi-gram ID LM is trained on the annotation of the training set as a regular LM.

5.4. Objective function for end-to-end learning

The objective function in learning matrix $\mathbf{W}$ is defined as the model-based expectation of slot-filling accuracy over the entire training set (proportional with a factor of N):

$$U(\mathbf{W}) = \sum_{n=1}^{N} \sum_{S_n} P_{\mathbf{W}}(S_n|O_n) C(S_n, S_n^*) \tag{9}$$

where N is the number of sentences in the training set, S_n^* is the slot ID sequence reference of the n-th input sentence O_n, and $S_n \in Hyp(O_n)$ that denotes the list of hypotheses of O_n. $C(S_n, S_n^*)$ is the accuracy measure of slot-filling, e.g., $C(S_n, S_n^*)$ could be the slot accuracy count or F-measure of S_n given the annotation reference S_n^*. Note that $C(S_n, S_n^*)$ is a measure irrelevant to parameter optimization. $P_{\mathbf{W}}(S_n|O_n)$ is the posterior defined in (5). The subscript $\mathbf{W}$ indicates that it is computed based on the parameter set $\mathbf{W}$ to be estimated.

5.5. Optimization

The objective function (9) has been widely used in a number of sequential pattern recognition tasks such as ASR (He et al. 2008) and SMT (He and Deng 2012). However, unlike previous work where the parameters to be optimized are in discrete probability distribution or continuous probability density domain, here we need to optimize an unbounded matrix. Therefore, the efficient EBW-based optimization method (He et al. 2008, He and Deng 2012) is not suitable anymore. Instead, we resort to the stochastic gradient descent (SGD) method to optimize $U(\mathbf{W})$.

By taking different forms of accuracy measure (S_n, S_n^*), the objective function $U(\mathbf{W})$ is directly linked to various evaluation metrics. Table 1 summarizes a few commonly used evaluation metrics for slot-filling and corresponding form of the accuracy measure. Note the corpus-level F-measure in the table is not decomposable. Therefore we use the average of sentence-level F-measure to approximate the corpus-level F-measure. In contrast, the corpus-level correct slot-ID count is directly decomposable to the sentence-level; i.e., it is simply the sum of correct slot-ID counts of all sentences.

Table 1. *Common evaluation metrics for SLU and their corresponding forms of the accuracy measure in Eq. (9).*

Metric	Accuracy measure $C(S_n, S_n^*)$
Concept error	Raw count of correct slot-IDs in S_n
F-measure	Sentence-level F-measure for S_n

6. EXPERIMENTS

6.1 Experimental Setup

In order to perform experiments with the DCNs, we compile a dataset of utterances from the users of a spoken dialog system. Table 1 shows the properties of the data sets and the (relative) frequencies of the two types of queries in each data set. Each of the utterances in these data sets is manually labeled with one of 25 domain categories. The domains were chosen to cover specific target domains such as restaurants, calendar, or movies, generic user intents such as greeting or frustration, and one *additional* category for the remaining domains. For evaluation, the error rate of the top scoring class is used. The baseline performance is obtained using only word trigrams with Boosting. This data contains about 125K word trigrams, so we applied Boosting-based filtering reducing the input feature space size to 4789 unique salient n-grams. These features are then fed to DCN and K-DCN.

We have also employed two additional kinds of features, following our earlier work (Hakkani-Tür et al 2012), using features as extracted from query click logs and named entity extraction. Both of these feature types have been shown to improve domain classification. Query click features are computed using the click distribution over a set of clicked base URLs (such as imdb.com and rottentomatoes.com) from search engine query click logs of user utterances. Some of these features are highly correlated as expected, as queries related to the same domain may be resulting in clicks to different base URLs related to that domain, and in this work these are represented as two different features. Since not every user utterance has these features are been observed in search query click logs due to the natural language nature of user utterances, which are different than keyword search queries, automatic translation of user utterances to keyword queries has been performed; see (Hakkani-Tür et al 2012) for more details on this translation.

6.2. Experimental Results

Shown in Table 1 are the summary results comparing the baseline system's performance with that of DCN and K-DCN systems. We

have used three types of raw features, including lexical features, features derived from query clicks, and features derived from name entities. Four ways of their combinations are fed into the three types of classifiers with the error rates shown in Table 1. K-DCN has considerably lower error rates than DCN and the baseline classifier for all four sets of raw features. Specifically, when query click features are added to the lexical features (row 3), all systems reduce error rates and the K-DCN system reduces the error rate more significantly. When name entity features, which take continuous values, are further added (row 4), the baseline system is not able to reduce its error rate, due to the difficulty of Boosting in handling non-binary features. Nevertheless, both DCN and K-DCN systems reduce their error rates, and for K-DCN the magnitude of error reduction is more significant again. One most interesting observation gleaned from Table 2 is that K-DCN is able to exploit query click features much more effectively than Boosting or DCN. While this is an area of further investigation, our preliminary observation is that K-DCN has the ability to directly handle mixed binary and continuous-valued inputs. In all experiments, no data normalization is carried out and the dynamic range of different elements of the input vectors can go from 0->1 to 0->50.

To provide more detailed results of the K-DCN in Table 1 with Lexical+QueryClick features, we show in Table 2 the domain classification error rates (percent) separately on Train set, Dev set, and Test set as a function of the depth or the module number (from bottom up) of the K-DCN. Importantly, the error rate of 5.94% is obtained at the lowest error rate of 6.45% for the Dev set, both occur at the sixth module of K-DCN. Beyond this, when more modules are stacked, the error rate of Dev and Test sets increasing, showing overfitting.

Table 2. *Comparisons of the domain classification error rates among the boosting-based baseline system, DCN system, and K-DCN system for a domain classification task. Three types of raw features (lexical, query clicks, and name entities) and four ways of their combinations are used for the evaluation as shown in four rows of the table.*

Feature Sets	Baseline	DCN	K-DCN
lexical features	10.40%	10.09%	**9.52%**
lexical features + Named Entities	9.40%	9.32%	**8.88%**
lexical features + Query clicks	8.50%	7.43%	**5.94%**
lexical features + Query clicks + Named Entities	10.10%	7.26%	**5.89%**

Table 3. *More detailed results of K-DCN in Table 2 with Lexical+QueryClick features. Domain classification error rates (percent) on Train set, Dev set, and Test set as a function of the depth of the K-DCN.*

Depth	Train Err%	Dev Error%	Test Err%
1	9.54	12.90	12.20
2	6.36	10.50	9.99
3	4.12	9.25	8.25
4	1.39	7.00	7.20
5	0.28	6.50	5.94
6	0.26	**6.45**	**5.94**
7	0.26	6.55	6.26
8	0.27	6.60	6.20
9	0.28	6.55	6.26
10	0.26	7.00	6.47
11	0.28	6.85	6.41

Note that, in Table 2, as the depth of K-DCN increases, the training error rate generally continues to decrease. If we were to select some fixed values of $\sigma^{(l)}$and $C^{(l)}$ over all modules l, the training error rate would go quickly to zero but overfitting would start earlier than module six to produce a higher error rate than 5.94%. To avoid this, we use a carefully tuned schedule for $\sigma^{(l)}$and $C^{(l)}$ as a function of l. This prevents the training error rate from quickly going to zero, thus causing overfitting to start after a low error rate has already been achieved.

We conduct slot-filling experiments on the ATIS dataset following similar settings as described in (Tur, et al., 2010). The training set consists of 4978 sentences and the test set consists of 893 sentences. In the task, each word will be tagged by a slot ID, and there is a total of 127 slot IDs. An example of a sentence and its annotations is given in section 3. The Linear CRF result is obtained using only lexical features, with default parameters of CRF++ toolkit, following (Raymond and Riccardi, 2007).

We then study the effectiveness of using K-DCN for local slot ID classification. In the experiment, we use a 5-word window for each position to derive raw features. We project each word to a 50-dimension dense vector by looking-up a embedding mapping table, which is trained through unsupervised learning on Wikipedia text corpus (Collobert, et al., 2011). Then we concatenate the 5 embedding vectors in a context window to form a 250-dimension contextual vector as the input for K-DCN. As in classical classification tasks, the output of K-DCN is a 127-dimension vector, each element corresponds to one slot ID. K-DCN is then trained on the training data. Compared to a logistic regression baseline which uses n-gram features (n=1~5) derived from the 5-word window, the K-DCN local classifier improves the F-measure significantly.

We then take the K-DCN output as dense local features for the log-linear model and perform the end-to-end training. A standard bi-gram LM on slot IDs is trained on the slot annotations of the training set. Then we performed SGD to train the softmax matrix by optimizing the expected F-measure. It gives further improvement by 0.23% over the K-DCN local classifier.

Table 4. S*lot-filling performances on ATIS.*

Models	F-measure
Logistic Regression	**90.07%**
Linear CRF	**91.09%**
K-DCN	**91.65%**
Log-linear K-DCN	**91.88%**

The gain from the end-to-end training is not as large as we expect, partially due to the reason that the slot dependency information is only modeled by a simple bi-gram LM, and it is not trained to optimize the end-to-end metric. In the future, we are working on extending the end-to-end sequential training methods to these bi-gram features and expect more significant results.

7. DISCUSSION AND CONCLUSIONS

This paper reports our ongoing research on the use of integrated deep learning and kernel learning for discriminative feature

extraction for SLU applications. The K-DCN architecture described in this paper can be viewed as an extension of the earlier DCN when the number of hidden units in each module grows to infinity. K-DCN is shown to perform much better than our previous boosting-based baseline and DCN systems when evaluated on a set of domain detection problems in SLU. For potential applications to slot filling problems of SLU, we make use of softmax to convert the discriminatively learned features computed by K-DCN into posterior probabilities in a log-linear model. An end-to-end learning technique is outlined to estimate the softmax weights that calibrate the K-DCN outputs so as to directly optimize the SLU performance metric. Experimental work in developing and testing the slot filling model is currently under way.

Compared with DCN, the K-DCN reported in this paper vastly increases the size of hidden units without suffering from computation and overfitting difficulties. However, as is typical of kernel methods, the memory required to hold the kernel matrix is quadratically related to the sample size, and when the sample size becomes very large, inverse of the correspondingly large matrix, as shown in Eq. (3), can become computationally expensive. For the SLU and speech recognition experiments involving larger training data than used in this study, our future work will develop feature selection/projection techniques, basis pursuit methods, and kernel approximation methods in the context of K-DCN (e.g., Baudat and Anouar, 2003; Cawley and N. Talbot, 2002). We will also strive to develop theory to guide the practice on cross-validation and adaptive regularization over modules of the K-DCN.

8. REFERENCES

[1] G. Baudat and F. Anouar. "Feature vector selection and projection using kernels," Neurocomputing, Vol. 55, pp. 22-38, 2003.

[2] C. Bishop. *Pattern Recognition and Machine Learning*, Springer, 2006.

[3] G. Cawley and N. Talbot, "Reduced rank kernel ridge regression," Neural Processing Letters, vol. 16, 2002.

[4] Z. Chen and S. Haykin, "On different facets of regularization theory," *Neural Computation* 14, 2791–2846, 2002.

[5] G. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pretrained deep neural networks for large vocabulary speech recognition," IEEE Trans. Audio, Speech, and Lang. Proc. Jan. 2012.

[6] L. Deng and D. Yu. "Deep Convex Network: A scalable architecture for speech pattern classification," Proc. Interspeech, 2011.

[7] L. Deng, D. Yu, and J. Platt. "Scalable stacking and learning for building deep architectures," Proc. ICASSP, 2012.

[8] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed, and G. Hinton. Binary coding of speech spectrograms using a deep auto-encoder. In Proc. Interspeech, September 2010.

[9] S. Hahn, M. Dinarelli, C. Raymond, F. Lefevre, P. Lehnen, R. de Mori, A. Moschitti, H. Ney, and G. Riccardi. "Comparing Stochastic Approaches to Spoken Language Understanding in Multiple Languages," IEEE TASLP, vol. 19, 2011.

[10] D. Hakkani-Tür, G. Tur, R. Iyer, L. Heck, "Translating Natural Language Utterances to Search Queries for SLU Domain Detection Using Query Click Logs," Proc. ICASSP 2012.

[11] X. He and L. Deng. "Maximum Expected BLEU Training of Phrase and Lexicon Translation Models", Proc. ACL, 2012.

[12] X. He, L. Deng, W. Chou. "Discriminative learning in sequential pattern recognition", IEEE Signal Processing Magazine, Sept. 2008.

[13] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition." IEEE Signal Processing Magazine, Vol. 29, No. 6, November, 2012.

[14] T. Hofmann and B. Scholkopf, and A. Smola, "Kernel methods in machine learning," The Annals of Statistics, Vol. 36, No. 3, pp. 1171-1220, 2008.

[15] G. Huang, H. Zhou, and X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," IEEE Trans. Systems, Man, and Cybernetics (Part B), Vol. 42, No. 2, pp. 513-529, April 2012.

[16] B. Hutchinson, L. Deng, and D. Yu, "A deep architecture with bilinear modeling of hidden representations: Applications to phonetic recognition," Proc. ICASSP 2012.

[17] H. Kadri, A. Rabaoui, P. Preux, E. Duflos, and A. Rakotomanonjy, "Functional regularized least squares classification with operator-valued kernels," Proc. ICML, 2011.

[18] Karasuyama, M. and Nakano, R. "Optimizing sparse kernel ridge regression hyperparameters based on leave-one-out cross-validation," *Proc. IJCNN 2008.* vol., no., pp. 3463-3468, June 2008.

[19] K. Macherey, O. Bender, H. Ney, "Applications of Statistical Machine Translation Approaches to Spoken Language Understanding", IEEE TASLP, vol. 17, 2009

[20] A. Mohamed, G. E. Dahl, and G. E. Hinton, "Acoustic modeling using deep belief networks," IEEE Trans. on Audio, Speech, and Lang. Proc. Jan. 2012.

[21] A. Mohamed, D. Yu, and L. Deng, "Investigation of full-sequence training of deep belief networks for speech recognition," Proc. Interspeech 2010, pp. 1692-1695.

[22] R. Monga, J. Dean, G. Corrado, K. Chen, M. Devin, Q. Le, M. Mao, M. Ranzato, P. Tucker, and A. Ng. "Downpour and Sandblaster: Two Complementary Approaches to Large-scale Distributed Optimization," ICML Workshop, 2012.

[23] F. Och. "Minimum error rate training in statistical machine translation", Proc. of ACL 2003.

[24] C. Raymond and G. Riccardi, Generative and Discriminative Algorithms for Spoken Language Understanding, Proc. Interspeech 2007.

[25] R. Rosipal and L. Trejo, "Kernel partial least squares regression in reproducing kernel Hilbert space," J. Machine Learning Research, vol. 2, pp. 97-123, 2001.

[26] C. Saunders, A. Gammerman, and V. Vovk, "Ridge regression learning algorithm in dual variables," Proc. ICML, 1998.

[27] G. Tur, L. Deng, D. Hakkani-Tür, and X. He. "Towards deep understanding: Deep convex networks for semantic utterance classification," Proc. ICASSP, 2012.

[28] G. Tur and R. De Mori (eds): *Spoken Language Understanding - Systems for Extracting Semantic Information from Speech*, John Wiley and Sons, 2011.

[29] G. Tur, D. Hakkani-Tür, and L. Heck. "What's Left to Be Understood in ATIS?," Proc. IEEE SLT Workshop, 2010.

[30] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa. "Natural Language Processing (Almost) from Scratch," Journal of Machine Learning Research (JMLR), 2011