Representation Learning for Information Extraction from Form-like Documents

Bodhisattwa Prasad Majumder[†♣] **Navneet Potti**[♠] **Sandeep Tata**[♠]
James B. Wendt[♠] **Qi Zhao**[♠] **Marc Najork**[♠]
[♣]Department of Computer Science and Engineering, UC San Diego
bmajumde@eng.ucsd.edu
[♠]Google Research, Mountain View
{navsan, tata, jwendt, zhaqi,
najork}@google.com

Abstract

We propose a novel approach using representation learning for tackling the problem of extracting structured information from form-like document images. We propose an extraction system that uses knowledge of the types of the target fields to generate extraction candidates, and a neural network architecture that learns a dense representation of each candidate based on neighboring words in the document. These learned representations are not only useful in solving the extraction task for unseen document templates from two different domains, but are also interpretable, as we show using loss cases.

1 Introduction

In this paper, we present a novel approach to the task of extracting structured information from form-like documents using a learned representation of an extraction candidate. Form-like documents like invoices, purchase orders, tax forms and insurance quotes are common in day-to-day business workflows, but current techniques for processing them largely still employ either manual effort or brittle and error-prone heuristics for extraction. The research question motivating our work is the following: given a target set of fields for a particular domain – e.g., due date and total amount for invoices – along with a small set of manually-labeled examples, can we learn to extract these fields from unseen documents?

Take, for instance, the domain of invoices, a document type that large enterprises often receive and process thousands of times every week (iPayables, 2016). Invoices from different vendors often present the same types of information but with different layouts and positioning. Figure 1 shows the headers of invoices from a few different vendors

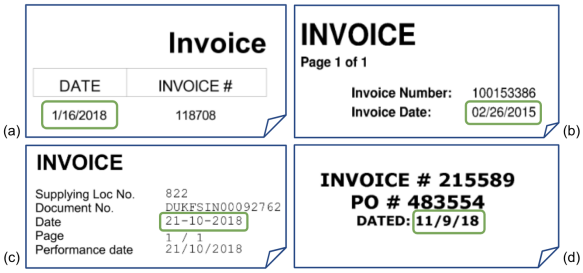

Figure 1: Excerpts from sample invoices from different vendors. Instances of the `invoice_date` field are highlighted in green.

showing the invoice date (highlighted in green) and number in different layouts. Furthermore, invoices from the same supplier even share similar presentation and differ only in specific values. We refer to this unit of visual pattern that is similar across a collection of documents as a *template*, and the fields of information that are common across templates in a domain as the *schema*. The schema consists of *fields* like `invoice_date` and `total_amount`, each associated with a *type* like `date` and `currency`.

Extracting values for these fields from a given document, particularly one belonging to an unseen template, is a challenging problem for many reasons. In contrast to most prior work on information extraction (Sarawagi, 2008), templatic documents do not contain much prose. Approaches that work well on natural text organized in sentences cannot be applied directly to such documents where spatial layout elements like tables and grid formatting are commonplace. Understanding spatial relationships is critical for achieving good extraction performance on such documents. Moreover, these documents are usually in PDF or scanned image formats, so these presentation hints are not explicitly available in a markup language. Techniques that are successful on HTML documents such as

[†] Work done during an internship at Google Research

web pages, including traditional wrapper induction approaches (Dalvi et al., 2011), are therefore not immediately applicable.

Recently, there has been a surge in research interest in solving this extraction task adapting techniques in natural language processing (Liu et al., 2019), computer vision (Davis et al., 2019), or combinations thereof (Katti et al., 2018). In contrast to this body of work, we propose an approach based on representation learning for this task. We first generate extraction candidates for each target field using its associated type (e.g., all dates as candidates for `invoice_date`). We then use a neural network model to learn a dense representation for each extraction candidate independent of the field to which it belongs. We also learn a separate representation for the field itself, and use the similarity between the candidate and field representations to score the candidate according to how likely it is to be the true extraction value for that field.

The design of our extraction system rests on a few observations about how information is often laid out in form-like documents (see Section 2). An advantage of our representation learning approach is that it allows us to encode certain priors we developed based on these observations into the architecture of the neural network and its input features (see Section 4). In fact, our experiments show that our proposed neural architecture outperforms a more naive MLP baseline using the same input features by about 10 F1 points on the extraction task for two different domains (see Section 6). Furthermore, the learned candidate representations are also meaningful and lend themselves to interpretation, as we show by delving into some loss cases.

2 Observations about Forms

We make three key observations about form-like documents that inform our design.

Observation 1 *Each field often corresponds to a well-understood type*. For example, the only likely extraction candidates for the `invoice_date` field in an invoice are instances of dates. A currency amount like $25.00 would clearly be incorrect. Since there are orders of magnitude fewer dates on an invoice as there are text tokens, limiting the search space by type dramatically simplifies the problem. Consequently, we use a library of detectors for several common types such as dates, currency amounts, integers, address portals, emails addresses, etc. to generate candidates.

Observation 2 *Each field instance is usually associated with a key phrase that bears an apparent visual relationship with it.* Consider the invoice excerpt in Figure 1(c). It contains two date instances, only one of which is the true `invoice_date`, as indicated by the word "Date" next to it. Similarly, in the bottom-right invoice excerpt, we are easily able to distinguish between the invoice number (indicated by "Invoice #") and the purchase order number (indicated by "PO #"). We call such indicative words *key phrases*.

Proximity is not the only criterion that defines a key phrase. For instance, the word "Date" is not the nearest one to the true `invoice_date` instance in Figure 1(c); the document number in the line above and the page number below are clearly closer. It is also not the case that the key phrase always occurs on the same line; Figure 1(a) shows a case where the key phrase "DATE" occurs just above the true `invoice_date`. An effective solution needs to combine the spatial information along with the textual information. Fortunately, in our experience, these spatial relationships exhibit only a small number of variations across templates, and these tend to generalize across fields and domains.

Observation 3 *Key phrases for a field are largely drawn from a small vocabulary of field-specific variants.* In a corpus of invoices we collected, we observed that, as exemplified by the samples in Figure 1, about 93% of the nearly 8400 invoice date instances were associated with key phrases that included the words "date" or "dated" and about 30% included "invoice". Only about 7% of invoice dates had neither of these words in their key phrases. Similarly, 87% of the nearly 2800 `due_date` instances in our corpus had key phrases that contained the word "due" and 81% contained "date". We found similar patterns for all other fields we investigated. The fact that there are only a small number of field-specific key phrases suggests that this problem may be tractable with modest amounts of training data.

While these observations are applicable to many fields across different document types, there are several exceptions which we plan to tackle in future work.

3 Extraction Pipeline

We leveraged the observations laid out in Section 2 to build a system to solve the information extraction task for form-like documents. Given a document

and a target schema, we generate extraction candidates for each field from the document text using the field type. We then score each candidate independently using a neural scoring model. Finally, we assign at most one scored candidate as an extraction result for each field. We discuss the stages of this pipeline here, and delve into the architecture of the scoring model in Section 4.

3.1 Ingestion

Our system can ingest both native digital as well as scanned documents. We render each document to an image and use a cloud OCR service[1] to extract all the text in it.

The text in the OCR result is arranged in the form of a hierarchy with individual characters at the leaf level, and words, paragraphs and blocks respectively in higher levels. The nodes in each level of the hierarchy are associated with bounding boxes represented in the 2D Cartesian plane of the document page. The words in a paragraph are arranged in reading order, as are the paragraphs and blocks themselves.

3.2 Candidate Generation

In Section 2, we made the observation that fields in our target schema correspond to well-understood types like dates, integers, currency amounts, addresses, etc. There are well-known techniques to detect instances of these types in text, ranging from regular expression matching and heuristics to sequence labeling using models trained on web data.

We associate each field type supported by our system with one or more candidate generators. These generators use a cloud-based entity extraction service[2] to detect spans of the OCR text extracted from the documents that are instances of the corresponding type. For example, every date in an invoice becomes a candidate for every `date` field in the target schema, viz. `invoice_date`, `due_date` and `delivery_date`.

Since the recall of the overall extraction system cannot exceed that of the candidate generators, it is important that their recall be high. Precision is, however, largely the responsibility of the scorer and assigner.

3.3 Scoring and Assignment

Given a set of candidates from a document for each field in the target schema, the crux of the extraction task is to identify the correct extraction candidate (if any) for each field. While there are many approaches one could take to solve this problem, we made the design choice to break it down to two steps: first, we compute a score $\in [0, 1]$ for each candidate independently using a neural model, then we assign to each field the scored candidate that is most likely to be the true extraction for it.

This separation of scoring and assignment allows us to learn a representation for each candidate based only on its neighborhood, independently of other candidates and fields. It also frees us to encode arbitrarily complex business rules into the assigner if required, for example, that the due date for an invoice cannot (chronologically) precede its invoice date, or that the line item prices must sum up to the total.

For brevity, we omit the details of the assignment module and report results using a simple assigner that chooses the highest-scoring candidate for each field independently of other fields.

4 Neural Scoring Model

The scoring module takes as input the target field from the schema and the extraction candidate to produce a prediction score $\in [0, 1]$. While the downstream assignement module consumes the scores directly, the scorer is trained and evaluated as a binary classifier. The target label for a candidate is determined by whether the candidate matches the ground truth for that document and field.

An important desideratum for us in the design of the scorer is that it learns a meaningful candidate representation. We propose an architecture where the model learns separate embeddings for the candidate and the field it belongs to, and where the similarity between the candidate and field embeddings determines the score.

We believe that such an architecture allows a single model to learn candidate representations that generalize across fields and document templates. We can conceptualize the learned representation of a candidate as encoding what words in its neighborhood form its associated key phrase since, apropos Observation 2, the spatial relationships between candidates and their key phrases are observed to generalize across fields. On the other hand, the embedding for a field can be conceptualized as encoding the key phrase variants that are usually indicative of it, apropos Observation 3.

[1] cloud.google.com/vision
[2] cloud.google.com/natural-language

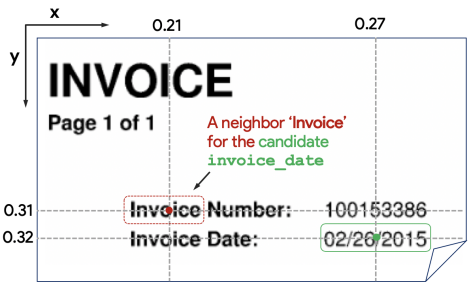

Figure 2: Neighbor 'Invoice' for `invoice_date` candidate with relative position $(-0.06, -0.01)$.

4.1 Candidate features

We would like our model to learn a representation of a candidate that captures its neighborhood. Accordingly, the essential features of a candidate are the text tokens that appear nearby, along with their positions. We use a simple heuristic to determine what OCR text tokens we consider to be the neighbors of a given candidate: we define a neighborhood zone around the candidate extending all the way to the left of the page and about 10% of the page height above it. Any text tokens whose bounding boxes overlap by more than half with the neighborhood zone is considered to be a neighbor.

As shown in Figure 2, we represent the position of a candidate and each of its neighbors using the 2-D Cartesian coordinates of the centroids of their respective bounding boxes. These coordinates are normalized by dividing by the corresponding page dimensions so that the features are independent of the pixel resolution of the input documents. We calculate the relative position of a neighbor as the difference between its normalized 2-D coordinates and those of the candidate. An additional feature we found to be helpful is the absolute position of the candidate itself.

An important design choice we made is to *not* incorporate the candidate text into the input. Note that this text was already the basis for generating the candidate in the first place. Withholding this information from the input to the model avoids accidental overfitting to our somewhat-small training datasets. For instance, since the invoices we collected were all dated prior to 2019, it is possible that providing the date itself as input to the model could cause it to learn that true `invoice_date` instances always occur prior to 2019.

4.2 Embeddings

As shown in Figure 3 (a)-(d), we embed each of the candidate features separately in the following ways.

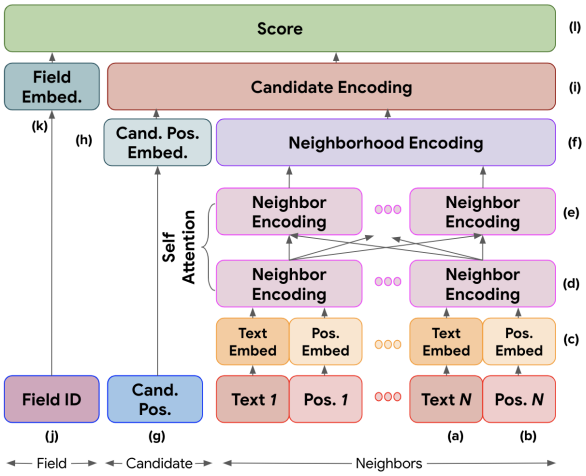

Figure 3: Neural Scoring Model. Pos. = Positional, Cand. = Candidate, Embed. = Embedding

The neighboring text tokens are embedded using a word embedding table. Each neighbor relative position is embedded through a nonlinear positional embedding consisting of two ReLU-activated layers with dropout. This nonlinear embedding allows the model to learn to resolve fine-grained differences in position, say between neighbors sharing the same line as the candidate and those on the line above. The candidate position feature is embedded using just a linear layer. We also use an embedding table for the field to which a candidate belongs.

In a model with embedding dimension d, the sizes of each neighbor's word and position embeddings are set to be d. We experimented with different sizes for the word and position embeddings, but it did not make a significant difference. For simplicity of exposition, we use the same value for both. Since each candidate is padded to have the same number of neighbors, say N, we denote the neighbor embeddings $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_N\}$, with each $\mathbf{h}_i \in \mathbb{R}^{2d}$. We also set the sizes of the candidate position embedding as well as the field embedding to be d.

Neighbor Encodings It is important to note that the initial neighbor embeddings $\mathbf{h}_i$ (Figure 3 (d)) are independent of each other. In order to capture interactions between neighbors, we employ *self-attention* (Vaswani et al., 2017), allowing each neighbor to have its embedding affected by all others. This is useful, for example, for the model to downweight a neighbor that has other neighbors between itself and the candidate.

We pack the neighbor embeddings $\mathbf{h}_i$ into a matrix $H \in \mathbb{R}^{N \times 2d}$, then transform these em-

bdeddings into query, key and value embeddings through three different linear projection matrices W_q, W_k and $W_v \in \mathbb{R}^{2d \times 2d}$.

$$\mathbf{q}_i = \mathbf{h}_i W_q \qquad K = HW_k \qquad V = HW_v$$

For each neighbor i, its query embedding $\mathbf{q}_i$ and the key embeddings K are used to obtain the attention weight vector $\alpha_\mathbf{i} \in \mathbb{R}^N$ as follows.

$$\boldsymbol{\alpha}_i = \text{Softmax}\left(\frac{\mathbf{q}_i K^T}{\sqrt{2d}}\right)$$

The self-attended neighbor encoding $\tilde{\mathbf{h}}_i \in \mathbb{R}^{2d}$ (see Figure 3(e)) for neighbor i is a linear combination of the value embeddings, $V \in \mathbb{R}^{N \times 2d}$, using the above attention weights for all the neighbors $\tilde{\mathbf{h}}_i = \boldsymbol{\alpha}_i V$.

As in Vaswani et al. (2017), we use a normalization constant of $\sqrt{2d}$ to improve stability. We project the self-attended neighbor encodings to a larger $4 \times 2d$ dimensional space using a linear projection with ReLU nonlinearity, and then project them back to $2d$.

4.3 Candidate Encoding

We combine the N neighbor encodings of size $2d$ each to form a single encoding of size $2d$ for the entire neighborhood. Since we already capture information about the relative positions of the neighbors with respect to the candidates in the embeddings themselves, it is important to ensure that the neighborhood encoding is invariant to the (arbitrary) order in which the neighbors are included in the features. Our experiments indicate that max-pooling the neighbor encodings together was the best strategy, slightly beating out mean-pooling.

Next, we obtain a candidate encoding (see Figure 3(f, h, i)) by concatenating the neighborhood encoding $\in \mathbb{R}^{2d}$ with the candidate position embedding $\in \mathbb{R}^d$ and projecting (through a ReLU-activated linear layer) back down to d dimensions.

Candidate Scoring The candidate encoding is expected to contain all relevant information about the candidate, including its position and its neighborhood. By design, it is independent of the field to which said candidate belongs. This neural network is, however, trained as a binary classifier to score a candidate according to how likely it is to be the true extraction value for some field and document.

Drawing inspiration from prior work in metric learning (Kulis, 2013), given a field with embedding $\mathbf{f} \in \mathbb{R}^d$ and its candidate with encoding $\mathbf{c} \in \mathbb{R}^d$, we compute $\text{CosineSimilarity}(\mathbf{c}, \mathbf{f}) \in [-1, 1]$. Finally, the model's prediction is simply a (constant) linear rescaling of this similarity so that the scores lie in $[0, 1]$. The model is trained using binary cross entropy between this prediction and the target label as the loss function.

Intuitively, this architecture ensures that the positive candidates for a field cluster together near its field embedding, and that these clusters are set far apart from each other. We use TSNE (Maaten and Hinton, 2008) to visualize this phenomenon in Section 6.2.

Corpus	Split	# Docs	# Templates
Invoices1	Train	11,390	11,390
	Validation	2,847	2,847
Invoices2	Test	595	595
Receipts	Train	237	141
	Validation	71	47
	Test	170	46

Table 1: Invoices and Receipts corpora

5 Datasets

To analyze the performance of our model, we used datasets belonging to two different domains, summarized in Table 1.

Invoices We collected two corpora of invoices from different sources. The first corpus, Invoices1, contains 14,237 single-page invoices. Each invoice was from a different vendor, so the documents do not share any common templates. Documents from the same vendor are generated from the same template. The second corpus, Invoices2, contains 595 documents belonging to different templates, with no templates in common with Invoices1. In all of our experiments, we used a 60-40 split of templates in Invoices1 as our training and validation sets, and all the templates in Invoices2 as our test set.

We asked human annotators to provide us ground truth extraction results for the fields shown in Table 2. The candidate generator associated with each field type was used to generate examples, which were then labeled using the ground truth.

About 95% of documents and fields present the training set had at least one positive example produced by our candidate generators. The field-level recall of our candidate generators varies from about 87% for `invoice_id` to about 99% for `invoice_date`. Improving the recall of candidate generators is part of our ongoing effort.

While the candidate generators have reasonably high recall, their precision varies dramatically from field to field. For common fields like `invoice_date` and `total_amount` that are present in nearly all documents, we generate fewer than ten negatives for each positive example. On the other hand, for rare fields like `total_tax_amount` as well as for fields with low-precision candidate generators such as the `alphanum` candidate generator for `purchase_order`, there can sometimes be dozens of negatives for each positive. Overall, since the negatives far outnumber the positives, we found it helpful to randomly downsample negatives in the training set to keep at most 40 negatives for each positive per field. The negatives in the validation and test sets were not downsampled.

We created a vocabulary of the 512 most frequent tokens, case-normalized, taken from the OCR text of the documents in Invoices1. The vocabulary also includes special tokens for numbers (`[NUMBER]`), out-of-vocabulary tokens (`[RARE]`) and padding (`[PAD]`). Despite the small size of this vocabulary, it covered at least 95% of words that occurred in key phrases across the entire corpus where excluded words were usually OCR errors.

Receipts We also evaluated our model using a publicly-available corpus of scanned receipts published as part of the ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction[3]. This corpus contains 626 receipt images with ground truth extraction results for four fields, viz., `address`, `company`, `date` and `total`. Using the `company` annotation as the template mapping, we found that these documents belong to 234 templates. The largest template contains 46 receipts and about half the documents belong to 13 templates with more than 10 documents each. On the other hand, nearly 70% of templates only have a single document. In all of our experiments, we used a 60-20-20 split of templates as our training, validation and test sets respectively, sampling at most 5 documents from each template.

Our target schema for this extraction task consists of the `date` and `total` fields. We generated labeled examples for these two fields using a vocabulary created as above from the 512 most frequent terms in the OCR text of the receipts. The fields in this dataset did not suffer from the label imbalance problem highlighted above for invoices.

[3] rrc.cvc.uab.es/?ch=13

6 Experiments

In this section, we evaluate our scoring model with respect to our two key desiderata. First, in Section 6.1, we show that our model is able to help the extraction system generalize to unseen templates. Then, in Section 6.2, we probe the model to show that it learns meaningful internal representations.

In the experiments described below, we trained models using the Rectified Adam (Liu et al., 2020) optimizer with a learning rate of 0.001 for 50 epochs. For both the Invoices and Receipts datasets described in Section 5, we used the training split to train the model, the validation split to pick the model with the best hold-out loss, and the test split to report performance metrics.

6.1 Generalization to unseen templates

We measured the performance of our model's scoring predictions using ROC AUC on the test split. We also analyzed its performance in the context of the overall extraction system using the accuracy of the end-to-end extraction results as measured by the maximum F1 score over all decision thresholds, averaged across all fields in the target schema shown in Table 2.

To demonstrate the benefits of our proposed neural architecture over a naive approach, we use two different baseline models for encoding a candidate and scoring it. The bag-of-words **BoW** baseline incorporates only the neighboring tokens of a candidate, but not their positions. The **MLP** baseline uses the same input features as our proposed model, including the relative positions of the candidate's neighbors, and encodes the candidate using 3 hidden layers. Both these baselines follow our representation learning approach, encoding the candidate and the field separately. Just as in our model, the final score is the cosine distance between the candidate and field encodings, normalized to $[0, 1]$ using a sigmoid.

We chose the dimension size for each model architecture using a grid-based hyperparameter search. All the metrics we report were obtained from performing 10 training runs and picking the model with the best validation ROC AUC.

Table 2 summarizes the results of this performance comparison. On both our evaluation datasets, our model showed a significant improvement over the baselines by both metrics. For the invoice corpus, our model outperforms the **BoW** baseline by about 1 point in the scorer ROC AUC,

Corpus	Field	Field Type	Train # +ves	Test % +ves	Scorer ROC AUC BoW	Scorer ROC AUC MLP	Scorer ROC AUC Ours	End-to-End Max F1 BoW	End-to-End Max F1 MLP	End-to-End Max F1 Ours
Invoices	amount_due	currency	5,930	4.8%	0.967	0.968	**0.973**	0.800	0.789	**0.801**
	due_date	date	5,788	12.9%	0.977	0.973	**0.984**	0.835	0.850	**0.861**
	invoice_date	date	13,638	57.4%	0.983	**0.986**	**0.986**	0.933	0.939	**0.940**
	invoice_id	alphanum	13,719	6.8%	0.983	0.988	**0.993**	0.913	0.937	**0.949**
	purchase_order	alphanum	13,262	2.2%	0.959	0.967	**0.976**	0.826	0.851	**0.896**
	total_amount	currency	8,182	12.5%	0.966	0.972	**0.980**	0.834	0.849	**0.858**
	total_tax_amount	currency	2,949	7.5%	0.975	0.967	**0.980**	0.756	0.812	**0.839**
	Macro-average	-		14.9%	0.973	0.974	**0.982**	0.842	0.861	**0.878**
Receipts	date	date	258	85.5%	0.748	**0.792**	0.737	**0.885**	**0.885**	0.854
	total	currency	475	16.7%	0.834	0.796	**0.889**	0.631	0.607	**0.813**
	Macro-average	-		51.1%	0.791	0.794	**0.813**	0.758	0.746	**0.833**

Table 2: Performance on the test set of unseen templates for Invoices and Receipts. The best-performing architecture in each case is **highlighted.**

which translates to about 3.6 points improvement in the end-to-end Max F1. In fact, our model beats the baseline in every field in our invoice target schema as well. This difference in performance clearly demonstrates the need to incorporate token positions to extract information accurately from form-like documents. Using neighbor position information, the **MLP** baseline is able to outperform the **BoW** baseline as well, but the improvement in end-to-end Max F1 is only about 2 points. This result demonstrates that our proposed architecture is better able to encode position information than a naive MLP.

Similarly, for the receipt corpus also, our model outperforms both the baselines. The improvement is much larger for the `total` field, more than 20 points. For the `date` field, since there are too few negative candidates in the dataset, all the models have comparable performance end-to-end.

A close examination of the per-field performance metrics in Table 2 reveals that model performance is greatly affected by both the number of positive training candidates, as well as by the ratio of positives to negatives. The best performance is observed for fields that occur frequently in invoices (e.g., `invoice_id`) and where the candidate generator emits only a small number of negatives for each positive (e.g., `invoice_date`). Conversely, the fields that are hardest to extract are those that are relatively rare and have low-precision candidate generators, viz., `amount_due` and `total_tax_amount`.

We also studied our model performance over various ablation setups and found that the relative order in which various features influence generalization performance is: neighbor text > candidate position > neighbor position. This result is also borne out by the fact that the **BoW** baseline, which omits the last of these features, is quite competitive with the other approaches.

We also compared the performance of our proposed architecture with and without the self-attention layer applied to the neighbor encodings. We found that self-attention contributes greatly to model performance for the invoice corpus: not only did self-attention lead to a 1-point improvement in scorer ROC AUC and a 1.7 point improvement in end-to-end max F1, we also observed an improvement in every single field in our invoice schema.

6.2 Meaningful internal representations

We investigated the internal representations learned by our model by visualizing their 2-D projections using TSNE. Figure 4(a) shows the representations learned for date candidates. They are colored based on the ground truth data indicating if they belong to one of `invoice_date`, `due_date`, or `delivery_date`. The learned encodings clearly show three distinct (by color) coherent clusters matching the respective field labels.

Figure 4(b) shows the candidate encodings for a sample of positive and negative date candidates for the `invoice_date` field, along with the embedding for that field. It is apparent that the encodings of the positive examples are largely clustered together whereas the sampled negatives show a more uniform and sparse spatial distribution. Furthermore, the field embedding lies close to the cluster of positive examples. It is interesting to note that the field embedding lies not at the center of the cluster, but rather at its edge, as far away as possible from the clusters of positive examples for other

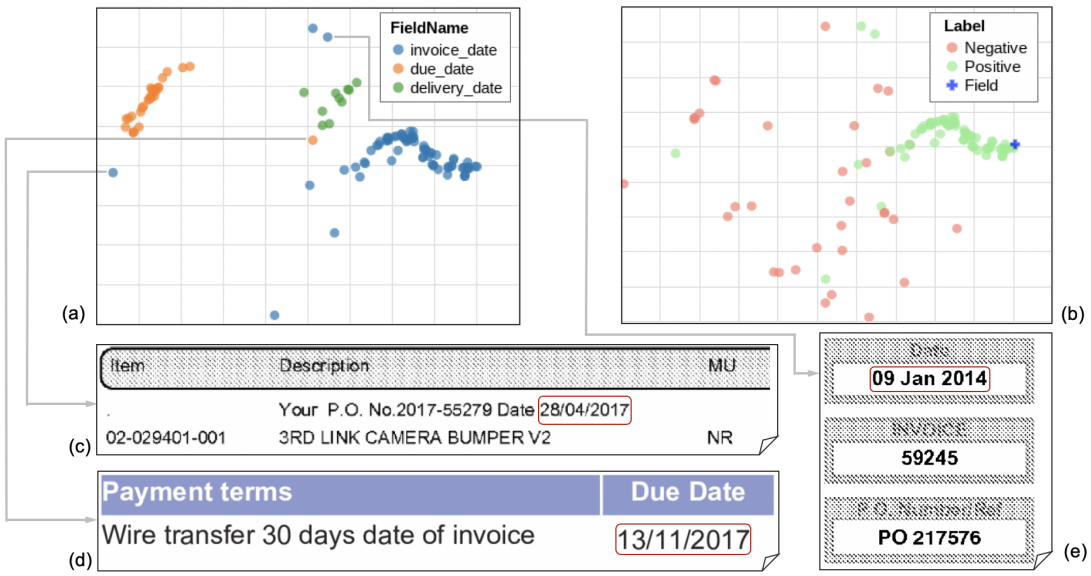

Figure 4: TSNE visualizations for (a) positive candidate encodings for the date fields in the target schema for invoices, and (b) positive and negative candidate encodings for `invoice_date` field as well as its field embedding. (c), (d) and (e) show three cases of misclustered candidate encodings

fields. This pattern is predicted by the fact that the loss function is essentially trying to minimize the cosine distance between the field embedding and its positives, while maximizing its distance from its negatives, most importantly the positives for the other fields.

We also indicate three cases of misclustered candidate encodings in Figure 4(a), whose corresponding invoice candidates and their neighborhoods are excerpted below. Figure 4(c) shows a ground truth positive `invoice_date` example whose encoding is far from the `invoice_date` cluster. It is clear from examining the invoice that this is an error in the ground truth labels provided by the human annotator. In fact, this date is the date of purchase and not the invoice date. The candidate shown in Figure 4(d) has a candidate encoding that lies midway between `due_date`, its true label, and `invoice_date`. We believe this is explained by the fact that this date has both the terms "Due Date" and "date of invoice" nearby, which are usually indicative of `due_date` and `invoice_date` respectively. Finally, Figure 4(e) shows a true `invoice_date` example whose encoding is far away from all the field clusters. A closer examination of the features of this candidate showed that our OCR engine was unable to detect the word "Date" just above the date due to scanning noise. Since this crucial word was missing from the neighbors of this candidate, the learned neighborhood representation was clearly incorrect.

7 Related Work

Information extraction from plain text documents for tasks like named entity recognition and relation extraction have benefited from recent advances in deep learning (Lample et al., 2016; Peng et al., 2017). However, these techniques are not directly applicable to our task on form-like documents. Palm et al. (2017) attempts to use RNNs to extract information from form-like documents. However, they treat each line as a vector of n-grams limiting the resulting accuracy.

The importance of understanding visual layout was recognized even in the context of information extraction of webpages in recent work (Cai et al., 2004; Yu et al., 2003; Zhu et al., 2006; Cai et al., 2003). The techniques developed by them are, however, not immediately applicable in our context since we do not have access to the source markup representation for the documents we deal with.

A common approach to solving the problem of extracting information from form-like documents is to register templates in a system, match new documents to an existing template, and use an extractor learnt from said template (Chiticariu et al., 2013; Schuster et al., 2013). The learning problem we tackle in this paper is more ambitious; we seek to generalize to unseen templates.

Our work is most closely related to recent attempts to combine layout features with text signals. Liu et al. (2019) use a document graph and intro-

duce a graph combination model to combine visual and textual signals in the document. Katti et al. (2018) represent a document as a two-dimensional grid of text tokens. Zhao et al. (2019) show that using grid information can be useful for information extraction tasks. Denk and Reisswig (2019) combine the grid-based approach with BERT-based text encodings. While an apples-to-apples comparison with these approaches is difficult without a shared benchmark, our system has several advantages: in contrast to the graph-based approaches (Liu et al., 2019) we focus on the harder problem of generalizing to unseen templates rather than dealing with the variations within a template. Since we are not starting with raw pixels, our approach is computationally less expensive than grid-based approaches. Further, we do not require clever heuristics to construct a multi-scale grid that is required for the image-segmentation style abstraction to work well.

To the best of our knowledge, our approach of using representation learning for this task is the first of its kind. We gain many of the well-known benefits of this approach (Bengio et al., 2013), most notably interpretability.

8 Conclusion and Future Work

In this paper, we presented a novel approach to the task of extracting structured information from templatic documents using representation learning. We showed that our extraction system using this approach not only has promising accuracy on unseen templates in two different domains, but also that the learned representations lend themselves to interpretation of loss cases.

In this initial foray into this challenging problem, we limited our scope to fields with domain-agnostic types like dates and numbers, and which have only one true value in a document. In future work, we hope to tackle repeated fields and learn domain-specific candidate generators. We are also actively investigating how our learned candidate representations can be used for transfer learning to a new domain and, ultimately, in a few-shot setting.

Acknowledgements We are grateful to Lauro Costa, Evan Huang, Will Lu, Lukas Rutishauser, Mu Wang, and Yang Xu on the Google Cloud team for their support with data collection, benchmarking, and continuous feedback on our ideas. We are also grateful for our research intern, Beliz Gunel, who helped re-run several experiments and fine-tune our training pipeline.

References

Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *TPAMI*, 35(8):1798–1828.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2003. Extracting content structure for web pages based on visual representation. In *Web Technologies and Applications, APWeb*, pages 406–417.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. 2004. Block-based web search. In *SIGIR*, pages 456–463.

Laura Chiticariu, Yunyao Li, and Frederick R. Reiss. 2013. Rule-based information extraction is dead! long live rule-based information extraction systems! In *EMNLP*, pages 827–832.

Nilesh Dalvi, Ravi Kumar, and Mohamed Soliman. 2011. Automatic wrappers for large scale web extraction. In *VLDB*, volume 4, pages 219–230.

Brian L. Davis, Bryan S. Morse, Scott Cohen, Brian L. Price, and Chris Tensmeyer. 2019. Deep visual template-free form parsing. *CoRR*, abs/1909.02576.

Timo I. Denk and Christian Reisswig. 2019. Bertgrid: Contextualized embedding for 2d document representation and understanding. *CoRR*, abs/1909.04948.

iPayables. 2016. Why Automation Matters: A Survey Study of the Modern Accounts Payable Department. Technical report, iPayables.

Anoop R. Katti, Christian Reisswig, Cordula Guder, Sebastian Brarda, Steffen Bickel, Johannes Höhne, and Jean Baptiste Faddoul. 2018. Chargrid: Towards understanding 2d documents. In *EMNLP*, pages 4459–4469.

Brian Kulis. 2013. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *NAACL*, pages 260–270.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. 2020. On the variance of the adaptive learning rate and beyond. In *ICLR*.

Xiaojing Liu, Feiyu Gao, Qiong Zhang, and Huasha Zhao. 2019. Graph convolution for multimodal information extraction from visually rich documents. In *NAACL*, pages 32–39.

Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *JMLR*, 9(Nov):2579–2605.

Rasmus Berg Palm, Ole Winther, and Florian Laws. 2017. Cloudscan - A configuration-free invoice analysis system using recurrent neural networks. In *ICDAR*, pages 406–413.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence n-ary relation extraction with graph lstms. *TACL*, 5:101–115.

Sunita Sarawagi. 2008. Information extraction. *Foundations and Trends® in Databases*, 1(3):261–377.

Daniel Schuster, Klemens Muthmann, Daniel Esser, Alexander Schill, Michael Berger, Christoph Weidling, Kamil Aliyev, and Andreas Hofmeier. 2013. Intellix - end-user trained information extraction for document archiving. In *ICDAR*, pages 101–105.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*, pages 5998–6008.

Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma. 2003. Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In *WWW*, pages 11–18.

Xiaohui Zhao, Zhuo Wu, and Xiaoguang Wang. 2019. CUTIE: learning to understand documents with convolutional universal text information extractor. *CoRR*, abs/1903.12363.

Jun Zhu, Zaiqing Nie, Ji-Rong Wen, Bo Zhang, and Wei-Ying Ma. 2006. Simultaneous record detection and attribute labeling in web data extraction. In *KDD*, pages 494–503.