End-to-end information extraction from business documents

Palm, Rasmus Berg

Publication date:
2019

Document Version
Publisher's PDF, also known as Version of record

Ph.D. Thesis
Doctor of Philosophy

DTU Compute
Department of Applied Mathematics and Computer Science

End-to-end information extraction from business documents

Rasmus Berg Palm

Kongens Lyngby 2018

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Richard Petersens Plads
Bygning 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

PHD-2018-501
ISSN 0909-3192

Summary

Extracting structured information from unstructured human communication is an ubiquitous task. There is a constant need for this task since computers do not understand our unstructured human communication, and we like to use computers to effectively organize our data. The study of performing this task automatically is known as Information Extraction (IE).

Current approaches to IE can largely be divided into two groups. 1) Rule based systems, which work by extracting information according to a set of pre-defined rules. These systems are flexible, and easy to understand but heuristic in nature. In addition the rules must be manually created and maintained. 2) Token classification systems, that works by classifying tokens, usually words, using machine learning. These are elegant and often superior to rule based systems, but require data labeled at the token level. This data is rarely available for IE tasks and must be explicitly created at great cost.

Inspired by the breakthroughs that end-to-end deep learning has had in several other fields, this thesis investigates end-to-end deep learning for information extraction. End-to-end deep learning works by learning deep neural networks that map directly from the input to the output data naturally consumed and produced in IE tasks. Since it learns from the data that is naturally available, for example as the result of a regular business process, it has the potential to be a widely applicable approach to IE.

The research papers presented in this thesis explore several aspects of end-to-end deep learning for IE. The main contributions are: 1) A novel architecture for end-to-end deep learning for IE, which achieve state-of-the-art results on a large realistic dataset of invoices. 2) A novel end-to-end deep learning method for structured prediction and relational reasoning. 3) A natural and efficient input representation of documents with combined text and image modalities.

Resume (Danish)

Ekstraktion af struktureret information fra ustruktureret menneskelig kommunikation er en allestedsnærværende opgave. Der er et konstant behov for denne opgave eftersom computere ikke forstår vores ustrukturerede menneskelige kommunikation og vi samtidigt ynder at bruge computere til at organisere vores data. Studiet af hvordan denne opgave kan udføres automatisk er kendt som Informations Ekstraktion (IE).

Nuværende tilgange til IE kan stort set deles op i to grupper. 1) Regelbaserede systemer, der virker ved at udtrække informationen i henhold til et sæt af foruddefinerede regler. Sådanne systemer er fleksible og nemme at forstår, men er heuristiske. Hvad mere er; reglerne skal laves og vedligeholdes manuelt. 2) klassifikationsbaserede systemer der virker ved at klassificere elementer, typisk ord, ved brug af machine learning. Disse systemer er elegante og ofte bedre end regelbaserede systemer, men kræver data der er annoteret på element niveau. Denne slags data er sjældent tilgængelig for IE opgaver og skal derfor laves eksplicit, til stor omkostning.

Inspireret af gennembruddene som ende-til-ende dyb læring har haft i adskillige andre felter, undersøger denne afhandling ende-til-ende dyb læring for informations ekstraktion. Ende-til-ende dyb læring virker ved at lære dybe neurale netværk som direkte beregner resultatet fra input af det data som naturligt er konsumeret og produceret i IE opgaver. Siden denne fremgangsmåde lærer direkte fra det data som naturligt er forekommende, f.eks. som resultatet af en normal forretningsprocess, har den potentiale til at være en bredt brugbar fremgangsmåde til IE.

Forskningsartiklerne som er præsenteret i denne afhandling undersøger flere aspekter omkring ende-til-ende dyb læring. Hovedbidragene er: 1) En ny arkitektur til ende-til-ende dyb læring for IE, som opnår gode resultater på et stort realistisk datasæt bestående af faktura. 2) En ny ende-til-ende dyb læring metode til struktureret prædiktion og relationel ræsonnering. 3) En naturlig og effektiv input repræsentation af dokumenter med kombineret tekst og billede modaliteter.

Preface

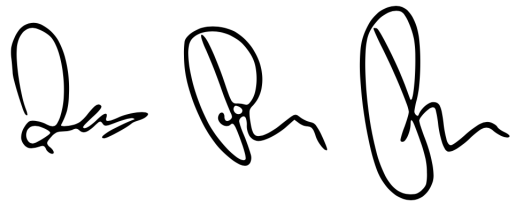
This thesis was prepared at the Cognitive Systems section of DTU Compute, department of Applied Mathematics and Computer Science at the Technical University of Denmark. It constitutes a partial fulfillment of the requirements for acquiring a Ph.D. degree at the Technical University of Denmark.

The Ph.D. project was sponsored by Tradeshift and the Innovation Fund Denmark under the industrial Ph.D. program, grant number 5016-00101B. It was supervised by Ole Winther of DTU Compute and Florian Laws of Tradeshift.

The work was carried out at Tradeshift and at the Cognitive Systems section of DTU Compute from November 2015 to October 2018, except for an external research stay at the University of Houston in the fall of 2017.

The thesis consists of four research papers.

Kongens Lyngby, October 31, 2018



Rasmus Berg Palm

Acknowledgements

First and foremost, I'd like to thank my supervisors Ole Winther and Florian Laws for all their support and encouragement these last three years.

I've been privileged to study alongside some great people. I'd like to thank Søren Sønderby, Lars Maaløe, Casper Sønderby, Marco Fraccaro, Rasmus Bonnevie, Simon Kamronn and Peter Jørgensen, for our stimulating conversations.

Thanks to Tradeshift and the Innovation Fund Denmark for funding the research.

Finally, a huge thanks to my lovely wife Helena Breth Nielsen, for her support throughout these studies.

Contributions

Included in Thesis

1. CloudScan - A configuration-free invoice analysis system using recurrent neural networks. *Rasmus Berg Palm, Ole Winther, Florian Laws* - Presented at International Conference on Document Analysis and Recognition (ICDAR) 2017.

2. End-to-End Information Extraction without Token-Level Supervision. *Rasmus Berg Palm, Dirk Hovy, Florian Laws, Ole Winther* - Presented at Workshop on Speech-Centric Natural Language Processing (SCNLP) at the Conference of Empirical Methods in Natural Language Processing (EMNLP) 2017.

3. Attend, Copy, Parse - End-to-end information extraction from documents. *Rasmus Berg Palm, Florian Laws, Ole Winther* - Unpublished.

4. Recurrent Relational Networks. *Rasmus Berg Palm, Ulrich Paquet, Ole Winther* - Accepted at Conference on Neural Information Processing Systems (NIPS) 2018.

Other contributions

1. Blayze - A fast and flexible Naive Bayes implementation for the JVM. *Rasmus Berg Palm, Fuyang Liu, Lasse Reedtz*
https://github.com/Tradeshift/blayze

Contents

CHAPTER 1

Introduction

1.1 The problem

This is an industrial Ph.D. project, focused on solving a single real-world problem. While this problem is an instance of a more general problem, and the research is relevant in this more general setting, it's instructive to explain the specific problem in some detail at this point. Many of the directions taken in the presented research stems from the specific constraints and challenges of this problem, and is easier to understand with a firm understanding of the specific problem.

An invoice is a document sent by a supplier to a buyer, listing the products or services bought, the debt owed, and when and how to pay said debt. Large companies can receive tens of thousands of invoices yearly. To efficiently manage such volumes they use software systems. The software systems can record when the invoices were received, manage payment, detect duplicate invoices, flag suspicious invoices, match invoices to orders, etc. All in all, greatly helping the company managing their purchases. Before any of these benefits can be enjoyed however, the information in the invoices must be extracted and entered into the software system. This is typically done by humans. However, this is a tedious and costly task, so an automated solution would be of great benefit.

The problem then is simple to describe: Turn the document image in figure 1.1 into the structured output in figure 1.2. That is, automatically extract a predefined set of important fields from an invoice. This automatic extraction forms the core of a Tradeshift product that help companies turn paper and PDF invoices into their structured, digital counterparts. You might argue that a PDF invoice is already a digital document. That is true. It is not, however, a *structured* document. There's no schema, detailing where each field must be, which format they must follow, etc. To a computer it's just a bunch of unstructured text and images. Contrast this with the structured version in figure 1.2. It follows a strict machine-readable schema and can be further processed in software system e.g. accounting and payment software as described above.

FAKTURA

Yderligere specifikation fremkommer ikke

Kunde: **Kunde nr.: 52200**

Rasmus Berg Palm
Sankt Hans Torv 3, 5. 3
2200 København N

i gården
25329691 /

Leverandør:

side 1/1

Peter Skafte ApS
Park Alle 352-A
2605 Brøndby
Telefon : 43 27 00 00
Swift : DABADKKK
E-mail : peter@skafte.dk
CVR Nr. : 14 29 04 00
IBAN : DK1230003557061940
Bank : 4440 3557061940
Site : www.skafte.dk

Faktura nr.: **722191** mandag 2. maj 2016

Vare nr	Varetekst	Antal	Vare a	Pant a	Beløb
8407	Leje Anlæg Tuborg Rå S	1,00	**300,00**	0,00	300,00
8904	Leje Kulsyre 1-4	1,00	**0,00**	0,00	0,00
8405	Fus Øko Tuborg Rå 25S	2,00	**612,00**	160,00	1.544,00
7774	Fadøls-krus 70x40cl	1,00	**76,00**	0,00	76,00
621	Kørsel lev+afh	1,00	**400,00**	0,00	400,00
875	Tom fus	-2,00	**0,00**	160,00	-320,00
	Sum af varer og pant				2.000,00
	moms 25%				500,00
	Fakturatotal til betaling			DKK	**2.500,00**

Betaling inden 06maj16 til Bank : 4440 3557061940
Rykkergebyr kr. 100,- + rente

Figure 1.1: An example invoice document image input.

```
{
  "number": "722191",
  "date": "2016-05-02",
  "order id": None,
  "buyer": "Rasmus Berg Palm",
  "supplier": "Peter Skafte ApS",
  "currency": "DKK",
  "sub total": 2000.00,
  "tax total": 500.00,
  "total": 2500.00,
  "tax percent": 25.00,
  ..., # additional fields
  "lines" : [
    {
        "id": "8407",
        "description": "Leje Anlæg Tuborg Rå S",
        "amount": 1.00,
        "price": 300.00,
        "total": 300.00
    },
    ... # the rest of the lines
  ]
}
```

.

Figure 1.2: The target structured output of the invoice in figure 1.1 as a python dictionary. Note many of the values does not appear verbatim in the inputs, e.g. the date, amounts, etc.

The dataset available comes from production usage of the above mentioned Tradeshift product. It consists of approximately 1.2 million pairs of PDF invoices, and their structured outputs, as seen in figure 1.1 and 1.2 respectively. The data has a single defining characteristic: The structured output data only denotes *what* the field values are, not *where* they are in the PDF document. If the data denoted where in the document the field values were, the problem would be a lot simpler. However, that is not the case, and this presents a challenge, which will shape much of the research. The reason the structured data only contain the value of each field, is because the users can freely type in the values for each field when using the product. The product will automatically suggest values for the fields, but if anything needs to be changed the user can simply type in the correct value. The product *could* require the users to specify where in the document a field value was, but this would place an additional labeling burden on the user, hindering them instead of helping them.

1.2 Deep Learning

The methods employed in this thesis all fall under the deep learning paradigm. As such I'll give a very brief and incomplete introduction here. For an excellent exposition see Goodfellow et al. [2016].

Conceptually deep learning is the belief that in order to solve complex data problems, the best approach is to use powerful deep machine learning models, trained on lots of data, which is represented as raw as possible [LeCun et al., 2015]. Contrast this with the alternative approach: designing complex features and using shallow classifiers. Deep and shallow refers to the number of learned functions that separate the input from the output. The problem with the latter approach is that it's hard for humans to design features that are descriptive and robust for certain problems, e.g. image recognition, speech recognition, etc. The deep learning approach instead *learns* to represent the inputs as a composition of multiple levels of intermediate representations. This is known as representation learning and is a key concept in deep learning [Goodfellow et al., 2016].

Conceptually deep learning is not tied to any specific machine learning model or learning algorithm. In practice however, it is deeply tied to neural networks and stochastic gradient descent. Neural Networks are a class of parameterized functions, and stochastic gradient descent is an optimization algorithm.

1.2.1 Optimization

Optimization is at the heart of deep learning, and machine learning in general. Assume you have a dataset of N input output pairs $\mathbf{x} = [x_1, ..., x_N]$ and $\mathbf{y} = [y_1, ..., y_N]$. In supervised learning the objective is to find a model, parameterized by θ, that maximize the probability of the outputs given the inputs, s.t.

$$\hat{\theta} = \max_{\theta} p(\mathbf{y}|\mathbf{x};\theta)\,. \tag{1.1}$$

If the data pairs can be assumed to be independent and identically distributed then $p(\mathbf{y}|\mathbf{x};\theta) = \prod_{i=1}^{N} p(y_i|x_i;\theta)$, s.t.

$$\hat{\theta} = \max_{\theta} \prod_{i=1}^{N} p(y_i|x_i;\theta)\,. \tag{1.2}$$

For numerical stability it's often convenient to minimize the negative log-probability instead of maximizing the probability directly,

$$\hat{\theta} = \min_{\theta} \sum_{i=1}^{N} -\log(p(y_i|x_i;\theta))\,. \tag{1.3}$$

The function that is being minimized is also denoted the loss function, $\mathfrak{L}(\theta; \mathbf{x}, \mathbf{y})$, since, loosely, it is a measure of how poorly the model models the data. In the above example $\mathfrak{L}(\theta; \mathbf{x}, \mathbf{y}) = \sum_{i=1}^{N} -\log(p(y_i|x_i; \theta))$.

One particular optimization algorithm, that is used extensively to optimize neural networks is stochastic gradient descent. Gradient descent works by iteratively computing the gradient of the loss function with respect to the parameters θ, and taking steps in the direction of the negative gradient until convergence or some other stopping criterion.

$$\Delta\theta_t = \frac{\delta\mathfrak{L}(\theta_t; \mathbf{x}, \mathbf{y})}{\delta\theta_t}\,, \tag{1.4}$$

$$\theta_{t+1} = \theta_t - \alpha\Delta\theta_t\,, \tag{1.5}$$

where α is a small scalar hyper-parameter, the step size, or learning rate. The partial derivatives of the parameters θ can be efficiently computed in neural networks using the back-propagation algorithm [Rumelhart et al., 1986]. If the dataset is large computing the loss function for the entire dataset can be prohibitively expensive. Stochastic gradient descent instead use a random sample of the dataset to approximate the loss function at every iteration. Under some mild constraints gradient descent converge to a local minima. If the loss function is convex this is also the global minima. However, deep learning often use highly non-convex functions, so there's no guarantees that gradient descent converges to the global minima. In practice it works well though. Why is an open research question [Dauphin et al., 2014, Dinh et al., 2017, Kawaguchi and Bengio, 2018].

1.2.2 Neural Networks

The basic unit of a neural network is the artificial neuron, a mathematical abstraction of the functioning of the biological neuron. The output of a single artificial neuron $a \in \mathbb{R}$ is

$$a = u\left(b + \sum_{i=1}^{I} \mathbf{w}_i \mathbf{z}_i\right)\,, \tag{1.6}$$

where $\mathbf{z} \in \mathbb{R}^I$ are the I inputs, $\mathbf{w} \in \mathbb{R}^I$ are I weights, $b \in \mathbb{R}$ is the bias and $u : \mathbb{R} \to \mathbb{R}$ is a non-linear function, typically the rectified linear unit $u(x) = \max(x, 0)$ [Nair and Hinton, 2010]. The entire neuron is a function $g_{\theta_g} : \mathbb{R}^I \to \mathbb{R}$ parameterized by $\theta_g = \{b, \mathbf{w}\}$ the scalar bias and weight vector.

A collection of J artificial neurons operating on the same input is known as a layer.

$$\mathbf{a}_j = u\left(\mathbf{b}_j + \sum_{i=1}^{I} \mathbf{W}_{ij} \mathbf{z}_i\right)\,, \tag{1.7}$$

where $\mathbf{a} \in \mathbb{R}^J$, $j \in [1, J]$ indexes the neuron in the layer and u is applied elementwise. The entire layer is a function $h_{\theta_h} : \mathbb{R}^I \to \mathbb{R}^J$ parameterized by $\theta_h = \{\mathbf{b} \in \mathbb{R}^J, \mathbf{W} \in \mathbb{R}^{I \times J}\}$ the bias vector and weight matrix.

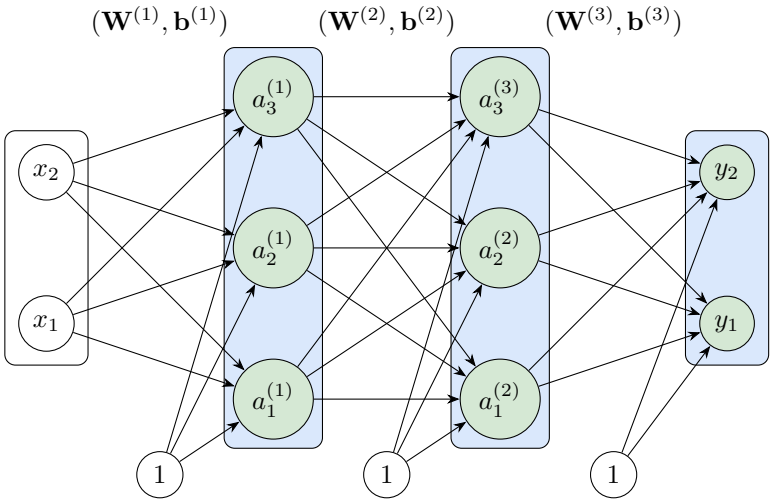


Figure 1.3: A fully connected neural network. The green circles are artificial neurons. The rectangles enclose layers. The blue background highlights layers with parameters $(\mathbf{W}^{(l)}, \mathbf{b}^{(l)})$.

Finally a neural network is a Directed Acyclic Graph (DAG) of layers. Let the neural network consist of L layers. let $J^{(l)} \in \mathbb{N}$ denote the number of neurons in layer l and let $\mathbf{a}_j^{(l)} \in \mathbb{R}$ denote the output of neuron j of layer l. The input vector to layer l, $\mathbf{z}^{(l)}$, is the concatenation of the output of the parent layers. As such the number of inputs to layer l, $I^{(l)} \in \mathbb{N}$, is the sum of the number of outputs of the parent layers: $I^{(l)} = \sum_{p \in P(l)} J^{(p)}$, where $P(l)$ is the set of indexes of parent layers for layer l.

$$\mathbf{a}_j^{(l)} = u^{(l)} \left(\mathbf{b}_j^{(l)} + \sum_{i=1}^{I^{(l)}} \mathbf{W}_{ij}^{(l)} \mathbf{z}_i^{(l)} \right), \tag{1.8}$$

where $u^{(l)} : \mathbb{R} \to \mathbb{R}$ is the non-linear function for layer l, $\mathbf{b}_j^{(l)} \in \mathbb{R}$ is the bias of node j in layer l, $\mathbf{W}_{ij}^{(l)}$ is the weight from input i to neuron j in layer l and $\mathbf{z}_i^{(l)}$ is the i'th element of the input vector $\mathbf{z}^{(l)}$ for layer l. The layers which does not have any parents, are the inputs to the neural network, typically denoted $\mathbf{x}$. The layers with no children are the outputs of the neural network, typically denoted $\mathbf{y}$. A neural network can also be described more generally without introducing layers, as a DAG of artificial neurons, but layers are a common abstraction. The entire neural network

is a function $f_{\theta_f} : \mathbb{R}^X \rightarrow \mathbb{R}^Y$ parameterized by $\theta_f = \{(\mathbf{b}^{(1)}, \mathbf{W}^{(1)}), ..., (\mathbf{b}^{(L)}, \mathbf{W}^{(L)})\}$ the parameters of the L layers, where X and Y are the dimensionality of the inputs and outputs respectively.

The description above introduces the vocabulary of neural networks and covers the typical architectures, but does not cover every neural network proposed. In the most general sense a neural network is just a parameterized function, that typically use artificial neurons. Two architectures deserve special mention: 1) the recurrent neural network and 2) the convolutional neural network.

Recurrent Neural Networks (RNNs) takes a sequence of vectors $[\mathbf{x}_1, ..., \mathbf{x}_T]$ as input, and computes a sequence of hidden state vectors $[\mathbf{h}_1, ..., \mathbf{h}_T]$ [Williams and Zipser, 1989]. The hidden states are defined recurrently:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}) \,, \tag{1.9}$$

where the function f determines how the recurrent neural network updates its internal state and $\mathbf{h}_0$ is predefined, typically either a vector of zeros, or a vector of learned parameters. See figure 1.4 for a graphical representation. The standard RNN defines $\mathbf{h}_t = \tanh(\mathbf{b} + \mathbf{W}_x \mathbf{x}_t + \mathbf{W}_h \mathbf{h}_{t-1})$, where $\mathbf{b}$, $\mathbf{W}_x$ and $\mathbf{W}_h$ is a bias vector, and two weight matrices respectively. The Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] and the Gated Recurrent Unit (GRU) [Chung et al., 2014] variants of f both help with long term memory of sequences and are popular choices. RNNs are powerful sequence models and have improved state-of-the-art in language modelling [Zaremba et al., 2014], machine translation [Wu et al., 2016], image captioning [Karpathy and Fei-Fei, 2015], speech recognition [Graves et al., 2013] and several other [Schmidhuber, 2015].

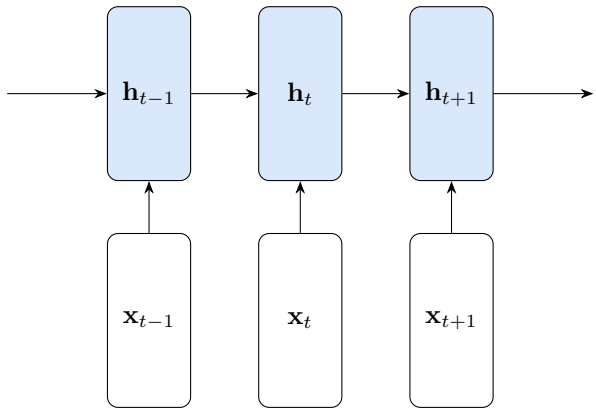


Figure 1.4: A recurrent neural network.

Convolutional Neural Networks (CNNs) were proposed for image recognition tasks [LeCun et al., 1989]. The key insight is that local features in images are translation invariant; a cat is still a cat, regardless of where it is in the image. Convolutional neural networks use artificial neurons, named "kernels", with small spatially local receptive fields, e.g. $5 \times 5 \times P$ pixels, where P is the number of channels in the input image, e.g. 3 for a RGB image. The output of kernel j in layer l, at position (x, y) is

$$a_j^{(l)}(x,y) = u\left(b_j^{(l)} + \sum_{p=1}^{P}\sum_{\delta_x=-\Delta_x}^{\Delta_x}\sum_{\delta_y=-\Delta_y}^{\Delta_y} \mathbf{W}_{p,\Delta_x+\delta_x,\Delta_y+\delta_y} a_p^{(l-1)}(x+\delta_x, y+\delta_y)\right), \tag{1.10}$$

where $u : \mathbb{R} \to \mathbb{R}$ is a non-linear function, $b_j^{(l)} \in \mathbb{R}$ is a bias term, P is the number of kernels in the previous layer, $(2\Delta_x + 1)$ and $(2\Delta_y + 1)$ are the input field size in the horizontal and vertical dimensions respectively, and $\mathbf{W} \in \mathbb{R}^{P\times(2\Delta_x+1)\times(2\Delta_y+1)}$ is the kernel weight tensor. Since the kernels are applied over the entire image, the number of parameters in a CNN is determined by the size and number of kernels, not the spatial size of the input. Compared to a fully connected neural network it "shares" parameters across spatial positions. The use of deep CNNs have lead to breakthroughs in object detection [Krizhevsky et al., 2012], face recognition [Taigman et al., 2014], image segmentation [Ronneberger et al., 2015], and *"ConvNets are now the dominant approach for almost all [Computer Vision] recognition and detection tasks"* [LeCun et al., 2015].

1.3 Information Extraction

The problem described is an instance of an Information Extraction (IE) task. Here I'll briefly give an overview of the field of information extraction and the approaches taken. The field of information extraction broadly studies extracting structured data from unstructured documents [Sarawagi et al., 2008]. Extracting such structured, machine-readable, information from unstructured human communication, is a task as mundane as it is ubiquitous. The constant demand for these tasks rests on three axioms. 1) People communicate using unstructured human language and artifacts (documents, emails, etc.), 2) people use computer systems for aggregating and managing information and 3) computers don't understand unstructured human communication. Imagine arranging a meeting with a co-worker. You send an email to your co-worker with the proposed date, room and time, and if she accepts, you create an event in your online calendar. Mapping back to the axioms, you: 1) communicated in human language 2) extracted the relevant information and typed it into a computer system and 3) your email client can't create the event for you, despite having all the information.

The grand prize would of course be that computers really *understood* unstructured

human language. However, it's likely that this essentially requires strong Artificial Intelligence (AI); an AI fully on-par with human intelligence in all aspects [Yampolskiy, 2013]. If we had such a system we could simply instruct it to extract the relevant information as we'd instruct a human and there would be no need for information extraction systems, or for a lot of other specialized machine learning systems for that matter. This has remained an elusive prize for decades however [Turing, 1950]. In the meantime IE takes the more pragmatic approach of automating small well defined information extraction tasks.

There's a variety of tasks considered in IE. The canonical task is template filling or event extraction; given a document extract information to fill a template. For instance an IE system could be built to extract information about terrorist attacks in newspaper articles. This was the main task type in the seminal Message Understanding Conference (MUC) series [Sundheim, 1991]. Such an IE system takes as input a newspaper article describing a terrorist attack and output a structured template of the terrorist attack detailing, e.g. the perpetrators, the victims, the date, etc. The problem considered in this thesis is an instance of this task. The latter MUCs introduced Named Entity Recognition (NER) and coreference resolution sub-tasks as well [Grishman and Sundheim, 1996]. Named Entity Recognition is the task of identifying proper nouns, e.g. named persons, organizations, places, etc. Coreference resolution is the task of finding expressions that refer to the same entity, e.g. resolving pronouns to their nouns, etc. The Automatic Content Extraction (ACE) research program replaced the MUC series and introduced the relation extraction task, which aims to extract generic relations, e.g. social relations (spouse, sibling, etc.), role relations (manager, staff, etc.), etc., from a corpus of documents [Doddington et al., 2004]. Finally the Text Analysis Conference (TAC) series succeeded the ACE program, and have focused on Knowledge Base Population (KBP), which aims to fill a knowledge base with facts from a corpus of documents. NER, coreference resolution and relation extraction can be seen as sub-tasks of KBP.

IE is closely related to the field of Natural Language Processing (NLP). NLP is *"... the subfield of computer science concerned with using computational techniques to learn, understand, and produce human language content"* [Hirschberg and Manning, 2015]. Since most IE is done on natural language text, IE often make use of methods originally developed for NLP.

Approaches to IE fall into two broad categories: 1) patterns and rules and 2) token classification.

1.3.1 Patterns and rules

There's typically some obvious patterns in the unstructured data and an intuitive approach is to try to use those patterns to extract the information. For instance,

if we wish to extract a date, we could look for strings that parse as dates, e.g. `"2018-02-23"`. Such dates could be captured with the following regular expression [Thompson, 1968] `/\d{4}-\d{2}-\d{2}/`, which will match strings of: four digits, dash, 2 digits, dash and 2 digits. This will match all ISO 8601 formatted date strings, but it'll also match `"9852-89-97"` and it won't match `"23rd Feb, 2018"`. We could extend this simple pattern, and create more patterns to try to cover all the various ways of writing dates, all the typical prefixes, e.g. "at", "on", "since", "after", etc. and all other relevant patterns we can think of. This would give us a set of patterns. Given a new document we would evaluate all our patterns on the document, which would give us a set of strings that matched one or more of the patterns. To decide which of these strings, if any, was the date, we would define a rule, e.g. "The date is the string matching most patterns and at least 5.". This date example is overly simple compared to actual research in this area. That is not to trivialize this area of research, but rather to let the reader appreciate the basic elements with a simple example.

Pattern and rule based IE has a rich history and is still being actively researched. Indeed a recent survey found that *"...rule-based IE dominates the commercial world..."* Chiticariu et al. [2013]. Here I'll only very briefly and non-exhaustively cover some of the more prominent ideas. Several systems make use of the syntactic structure of sentences, e.g. the terrorist target is the subject in sentences with the "bombed" past tense verb [Riloff et al., 1993, Huffman, 1995, Soderland et al., 1995]. Kim and Moldovan [1995], Soderland et al. [1995] also considers the semantics, e.g. only match targets that are buildings. To alleviate the need for manually writing these patterns and rules, several works propose automatically extracting instances of patterns given broad classes of patterns and a few labeled examples [Ciravegna, 2001, Califf and Mooney, 2003, Gupta and Manning, 2014a]. Since building a rule based IE system is often a highly iterative process of creating and adjusting rules, another vein of research focuses on improving interfaces for visualizing and debugging rules [Gupta and Manning, 2014b, Liakata et al., 2009]. See Muslea et al. [1999] and Patwardhan [2010] for surveys on rule based IE.

In general the main strength of rule based IE systems is that they are interpretable and easily modified [Chiticariu et al., 2013]. Errors can be traced back to rules and rules can be understood and modified by users. They are also often flexible, allowing users to quickly create rules and patterns for a new IE task based on predefined classes of patterns. The main drawback is the need to manually create and adjust the rules and that the systems are heuristic in nature, i.e. there's no guarantee that the decisions are optimal in any sense [Chiticariu et al., 2013]. There's also a risk that the rules become brittle, e.g. minor spelling errors or formatting changes can throw them off. Rule based IE works better the more homogeneous and structured the input is. For instance, rule based IE works very well for extracting information from well defined forms such as immigration forms, tax returns, etc.

For the specific task of extracting information from invoices, several systems have been proposed that are based on patterns and rules [Esser et al., 2012, Rusinol et al., 2013, Cesarini et al., 2003, Dengel and Klein, 2002, Schuster et al., 2013, Medvet et al., 2011]. These systems generally assume that invoices from the same supplier follow the same layout or template. In broad terms they work by letting the user define a set of rules for each supplier. These are either hand-written or induced from broad classes of patterns by labeled examples. These rules are typically based on keywords, absolute and relative positions and regular expressions. For instance, a rule might be that the first word to the right of the word `"total"` which can also be parsed as an amount is the total amount. Given a new invoice the systems first classify the supplier of the invoice, then apply the rules for that supplier. The rules suggests strings to be extracted which are then scored using heuristics, e.g. whether the totals found add up, etc.

1.3.2 Token classification

Token classification is the other major approach to information extraction. The idea is to classify which tokens, typically words, to extract using supervised machine learning. It requires that each token is labeled. The Air Travel Information Services (ATIS) dataset is a good example [Price, 1990]. It consists of 5871 transcribed airfare queries with each word in the query labeled with one of 127 labels using Beginning-Inside-Out (BIO) labels [Ramshaw and Marcus, 1995]. See table 1.1 for an example. The BIO labeling scheme enables the extraction of fields that span multiple word. The beginning prefix, `"B-"`, denotes the beginning of a label, and subsequent inside prefixes, `"I-"`, denotes a continuation of that label. The outside label `"O"` denotes that the word is none of the labels. Given the labeled sentence in table 1.1 a ”chunking” algorithm would iterate over the words, joining contiguous label segments and output `"cost_relative": "cheapest"`, `"toloc.city_name": "new york city"`, etc.

word	label
cheapest	B-cost_relative
flight	O
to	O
new	B-toloc.city_name
york	I-toloc.city_name
city	I-toloc.city_name
from	O
la	B-fromloc.city_name
tomorrow	B-depart_date.today_relative

Table 1.1: ATIS dataset example.

Given such a dataset, supervised machine learning can be used to learn a classifier that classifies the labels for each word. Consider the case where you have K words and label pairs $[(x_i, y_k)..., (x_K, y_K)]$ for a single document. Let $\mathbf{x} = [x_1, ..., x_K]$ denote all the feature vectors, and $\mathbf{y} = [y_1, ..., y_K]$ all the labels. The feature representation, x_i, of a word is critical for the performance of any supervised learning classifier. Typical features include one-hot encoding the word or using a learned vector embedding [Mikolov et al., 2013], whether the word is present in lists of known keywords, e.g. cities, names, etc., syntactic information, e.g. part of speech tags and finally whether the word matches various regular expressions. Most rules from the rule and pattern based approach to information extraction can also be reformulated as binary features, whether the word would have been extracted by the rules.

The next step is to choose a classifier. An important distinction is how the classifier handle the context of each word. The simplest classifiers treat the labels as conditionally independent given the feature vectors, s.t. $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{K} p(y_i|x_i)$. Any standard supervised classifier can be used e.g. logistic regression, neural networks, decision trees, Support Vector Machines (SVMs) [Hearst et al., 1998], etc. In this case it's important that the features capture enough context to classify the words. Alternatively a fixed size context window can be used s.t. $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{K} p(y_i|x_{i-M}, ..., x_i, ..., x_{i+M})$, which would correspond to a window of size $2M+1$. For the indices where the window fall outside the sequence, i.e. $i < M + 1$ and $i > K - M$ padding with empty feature vectors are typically used. Convolutional Neural Networks naturally fall into this category. Also, all the same classifiers as before can be used since the $2M + 1$ feature vectors can be seen as a single feature vector with $2M + 1$ the amount of entries. The next step is the classifiers which map a variable number of feature vectors into a fixed size representation h_i and then consider the labels conditionally independent given h_i, i.e. $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{K} p(y_i|h_i)$. Recurrent Neural Networks fall into this category, using a recurrent function definition $h_i = g(x_i, h_{i-1})$ which maps the feature vectors of the current and all previous words into a fixed size h_i. Given pre-computed h_i any of the standard classifiers can also be used. The final set of classifiers also take the dependence between the labels into account. An example is linear chain Conditional Random Fields (CRFs) which models the pairwise dependence between labels s.t. $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{i=1}^{K} \psi_u(y_i, \mathbf{x}) + \sum_{i=1}^{K-1} \psi_p(y_i, y_{i+1}, \mathbf{x})\right)$, where ψ_u and ψ_p are functions that map to $\mathbb{R}$, and $Z(\mathbf{x})$ is the partition function.

The main advantages of the token classification approach are that 1) the supervised learning algorithm can find complex patterns in the features that a human might not be able to define and 2) the discovered patterns are learned from the training data, which means they are robust to common noise present in the training data, e.g. common misspellings, formatting differences, etc. Also, given relatively simple features a sufficiently powerful classifier can automatically discover patterns from complex combinations of the features. This shifts the burden of designing complex patterns and rules from the human to the learning algorithm. The main drawbacks are

1) the extensive labeling effort needed and 2) the learned classifier is often opaque, i.e. it's hard to understand how it works, which can make it hard to improve [Chiticariu et al., 2013]. Due to the extensive labeling required for token classification, acquiring it is costly and time-consuming. As such, this approach is not feasible for many real life IE tasks.

CHAPTER 2

Research

2.1 Motivation

Deep Learning have lead to breakthroughs in several fields. Many of the key breakthroughs have come when researchers discovered efficient ways of learning to solve the complete problem end-to-end, instead of breaking it down and trying to solve sub-problems. Manually creating features for object classification gave way to learning object recognition end-to-end [Krizhevsky et al., 2012]. Machine translation started as translating words, then phrases, to finally being trained on whole sentences [Wu et al., 2016]. Text-to-speech went from concatenating small units of speech and using vocoders to directly outputting the raw waveform [Van Den Oord et al., 2016].

Inspired by such breakthroughs this thesis explores the use of end-to-end deep learning models for the problem of information extraction from invoices. How can the information extraction task be formulated and solved in an end-to-end manner? This end-to-end approach represents a third approach to information extraction, distinct from the rule based and token classification approaches discussed.

The research is inspired by the breakthroughs mentioned, but also by necessity; the available data from Tradeshift is simply of an end-to-end nature. I don't believe this is a special case, rather I think it represents the vast majority of information extraction tasks currently performed by people. Simply by performing the tasks they are generating end-to-end data; unstructured data in, structured data out. The specifics of the generated data varies, but will have one thing in common: it is not created for machine learning purposes. Data that can be used for machine learning, e.g. token labels, can of course be explicitly created, but this will invariably incur extra costs. End-to-end information extraction concerns itself with learning from the available data directly. This make it applicable for tasks that would otherwise require either 1) a custom rule based system or 2) expensive labeling.

2.1.1 The general case for end-to-end

Breaking down large problems and solving the smaller sub-problems is a powerful strategy, and deeply ingrained in engineering and computer science. This is how we

build everything from houses to operating systems. So why doesn't it work for object detection, machine translation, speech synthesis, etc.? What's the case for end-to-end learning systems in general?

False assumptions of independence Breaking a problem down often requires assumptions of independence which can lead to irreducible errors. As an example consider state-of-the-art machine translation systems which are trained on corpus of aligned sentences [Wu et al., 2016]. This assume that sentences can be translated independently of each other, which is not always the case. Take for instance the sentences pairs "He liked fishing. Especially bass." and "He liked playing instruments. Especially bass.". The latter sentences are the same, but should have different translations. If the model translates the sentences independently those type of errors cannot be reduced. Further, errors introduced in this way add up. If a problem is broken into 10 sub-problems, which each introduce 2% irreducible, independent errors, then the joint solution will have approximately 18.3% irreducible errors.

More data Another possible reason is the hypothesis that there is more data available for the real problems. Sub-problems have less data since they are often not worthwhile solving in and of themselves. As such datasets for sub-problems often have to be explicitly created, at considerable cost. For instance, solving machine translation at the word, phrase or sentence level requires alignments at the respective level, i.e. these N words in the source text correspond to these M words in the target text. This data is not valuable or useful for anything except creating machine translation models. Contrast this with the proceedings of the European Union which have been translated into 21 member languages [Koehn, 2005]. This large dataset is available because these translation are inherently valuable to the European Union. This is also the case for the invoice data from Tradeshift. It is created because it is valuable in its own right, not because it is valuable for machine learning purposes.

2.2 Approach

All models that can transform input into output in an end-to-end manner are not equal. If this was the case we could simply treat every supervised problem as a sequence of bytes in and a sequence of bytes out and use sequence to sequence models for everything. The model and data representations have to be, for lack of a better word, *efficient*. I don't have a clear definition of efficient, but I'll give some examples of what I mean. The output representation should be as simple as possible. For instance, instead of representing a currency as a three letter code, e.g. "USD", it should be treated as a classification problem with approximately 180 classes corresponding to the different currencies in the world. The former representation has $26^3 = 17,567$ possible states of which only 180 are valid. In other words, it should be as hard as possible for the model to make a mistake. Similarly the input should be represented

as "naturally" as possible. For instance, in Palm et al. [2017b] we represent the words in the invoice as a sequence, read left to right, top to bottom. This is not optimal since it ignores the spatial layout of the words, e.g. columns, etc. Lastly, hard independence assumptions made when breaking down a problem should instead be incorporated as soft, structural priors. Take for instance machine translation, in which the major breakthrough came when attention was introduced, which acts as a learned, soft, word-alignment model [Bahdanau et al., 2014]. Similarly fully connected neural networks *can* learn exactly the same functions that CNNs can, but the structural priors incorporated in CNNs make them much better. The trick is making the models just flexible enough and offering shortcuts and hints in the form of structural priors where possible.

Finding the right input and output representations, and finding the architecture with the right structural priors that make end-to-end training feasible and efficient represent the bulk of the research presented here.

The research presented in this thesis does not consider all aspects of the problem. Three major areas are not considered:

1. Converting images of text characters into machine-encoded text, known as Optical Character Recognition (OCR). Very good OCR engines already exists and this is a whole area of research in itself. The presented research assumes an OCR engine is used to extract the text.

2. Extracting the lines. The lines are important, but I decided to consider the problem of the header-level fields first, as I reasoned that any solution for extracting the lines would build upon a solid understanding of extracting individual fields. After all, a line is simply a collection of fields.

3. Some of the fields to be extracted have a limited output space, e.g. there's only around 180 currencies in the world. Similarly, there's a known set of companies using Tradeshift representing the possible buyers and suppliers. Extracting the values for these fields can be formulated as classifying the entire invoice into one of the possible outputs. This is known as document classification, and is another well studied research field. The research presented here only consider the harder task of the fields with an effectively unlimited output space, e.g. the amounts, the invoice number, etc.

The following sections summarize the research papers, discuss how they fit into the overall research goals and how they fit together.

2.3 Papers

2.3.1 CloudScan - A configuration-free invoice analysis system using recurrent neural networks

***Rasmus Berg Palm**, Ole Winther, Florian Laws* - Presented at International Conference on Document Analysis and Recognition (ICDAR) 2017. See appendix A.

Palm et al. [2017b] introduce the problem, argues for a configuration-free approach and present a token classification model. The focus on configuration-free should be seen in contrast to the other state-of-the-art systems for extracting information from invoices. These systems are rule-based systems, and thus require configuration before they can be used for a new supplier, typically in the form of labeling. The proposed system is based on token classification, and generalize across suppliers. As such a new supplier can use the system with zero up-front configuration. Learning a single model that generalize across suppliers and invoice formats was a significant step from rule-based systems towards a learned end-to-end system. The proposed system works by classifying N-grams in the invoice into one of 65 classes (32 classes using BIO notation). Using these classifications a set of heuristics picks the words for each field. See figure 2.1 for an overview and table 2.1 for the results on unseen invoice layouts.

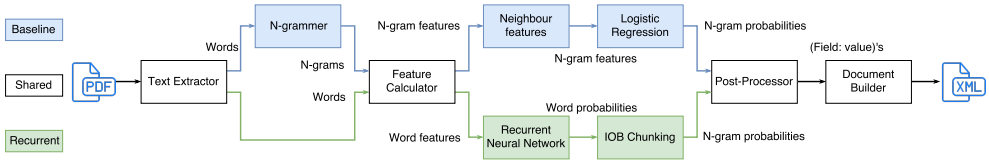


Figure 2.1: The two systems evaluated. The top (blue) track is the baseline, and the bottom (green) track is the recurrent neural network based system. White components are shared between the two systems.

The proposed system has a major drawback. It requires data labeled at the token level. Since this data is not available the paper propose to infer it from the available end-to-end data. In short the inference procedure works by noting that if the total is "2500.00", then any word in the PDF that can be parsed to "2500.00" gets the label "total". This is closely related to the idea of distant supervision by Mintz et al. [2009]. Depending on the recall and precision of the parsers this procedure will miss legitimate totals, and introduce spurious ones. This introduces noise in the training and testing data. We hypothesized that the noise would be relatively random compared to the correct words, such that a classifier, with the right amount of regularization, might learn to recognize the right words, and ignore the noise. Regardless, it's inelegant, and hard to optimize since at some point, doing better on the evaluation set means

	F1		Precision		Recall	
Field	Baseline	LSTM	Baseline	LSTM	Baseline	LSTM
Number	0.71	**0.760**	0.76	**0.79**	0.67	**0.73**
Date	0.69	**0.774**	0.76	**0.85**	0.64	**0.71**
Currency	**0.91**	0.91	0.98	**0.98**	**0.85**	0.84
Order ID	0.43	**0.52**	**0.82**	0.74	0.29	**0.41**
Total	0.84	**0.90**	0.86	**0.91**	0.82	**0.88**
Line Total	0.80	**0.88**	0.83	**0.89**	0.78	**0.87**
Tax Total	0.83	**0.88**	0.84	**0.88**	0.83	**0.87**
Tax Percent	0.81	**0.87**	0.83	**0.89**	0.80	**0.85**
Micro avg.	0.79	**0.84**	0.84	**0.88**	0.75	**0.80**

Table 2.1: Expected performance on next invoice from unseen template. Best results in bold. From Palm et al. [2017b].

picking up more of the noise. A further complication is that the actual measure of performance we're interested in, is the end-to-end performance. Since the values for each field are ultimately chosen by the heuristics, a better token classifier does not always translate into better end-to-end performance.

How to represent the input such that the the text and image modality was combined in a natural way was a major research question throughout the thesis. In this paper we ignore the image modality and propose to represent the words in the invoice as a sequence, read left to right, top to bottom. We show how this representation is better at capturing the context of a word than using a fixed context window of the four closest words in the cardinal directions. Regardless, it's an inelegant solution since it ignores the spatial layout of the words, and forces us to impose a somewhat spurious ordering.

2.3.2 End-to-End Information Extraction without Token-Level Supervision

***Rasmus Berg Palm**, Dirk Hovy, Florian Laws, Ole Winther* - Presented at Workshop on Speech-Centric Natural Language Processing (SCNLP) at the Conference of Empirical Methods in Natural Language Processing (EMNLP) 2017. See appendix B.

Palm et al. [2017a] introduces the end-to-end information extraction task in a simplified setting. The task is simplified by using standard text based IE datasets which

are labeled with BIO labels at the token level, e.g. ATIS. From these labels we derive equivalent end-to-end datasets. This simplifies the task in two ways. First, the input is regular text, so there is no document image to consider. Second, the label values in the end-to-end dataset are always present, verbatim, in the input text. Since the original datasets are labeled at the token level we can compare to state-of-the-art sequence classification methods. We propose an end-to-end neural network model based on pointer networks [Vinyals et al., 2015] for extracting the information and show that it is comparable to state-of-the-art neural sequence classification models. See figure 2.2 for an overview and table 2.2 for the results.

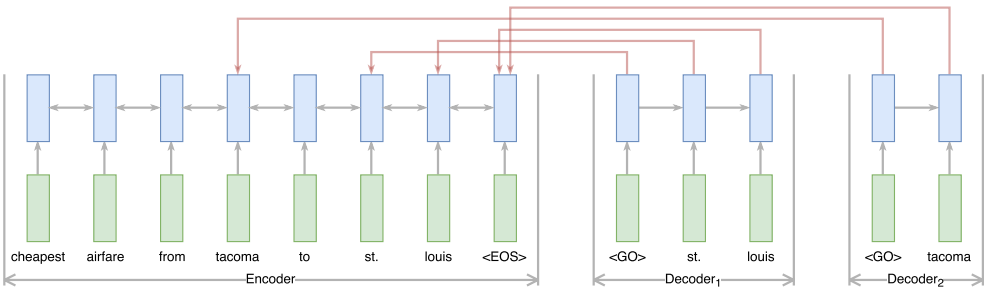


Figure 2.2: The multi-head pointer network. At each step each decoder "points" to a word in the encoded inputs, indicated by the red arrows. From Palm et al. [2017a].

The main drawback of the proposed model is that it can only output words that are present in the input text. This means that if the target output is not verbatim in the input the network will always fail. For the general end-to-end task it is likely that values in the output will not match values in the input verbatim. For instance, if the input contains the date "16 Oct. 2018" the structured output would contain the date in some standard format, e.g. ISO 8601 "2018-10-16". Despite its shortcomings the paper made clear what the remaining obstacle was to achieve end-to-end training: figuring out how to parse the extracted input into normalized output.

Data set	Baseline	Ours	p
ATIS	0.977	0.974	0.1755
Movie	0.816	0.817	0.3792
Restaurant	**0.724**	0.694	0.0001

Table 2.2: Micro average F1 scores on the E2E data sets. Results that are significantly better ($p < 0.05$) are highlighted in bold. From Palm et al. [2017a].

2.3.3 Attend, Copy, Parse - End-to-end information extraction from documents

***Rasmus Berg Palm**, Florian Laws, Ole Winther* - Unpublished. See appendix C.

Palm et al. [2018a] address the end-to-end information extraction task directly and make substantial progress. First, the text and image modalities are combined in a natural way by concatenating learned word embeddings with the document image as extra image channels at the positions of the embedded words. The input "image" thus have $3 + D$ channels, where the first three channels are the red, green, blue image channels and D is the dimensionality of the learned word embeddings. The word embeddings are replicated across the spatial extent of each word in the document image. This is a quite elegant representation of the input in my opinion since it naturally links the word features and the image features. A CNN working on this input can discover the importance of spatial structure itself, without requiring us to impose any spurious ordering on the words, etc.

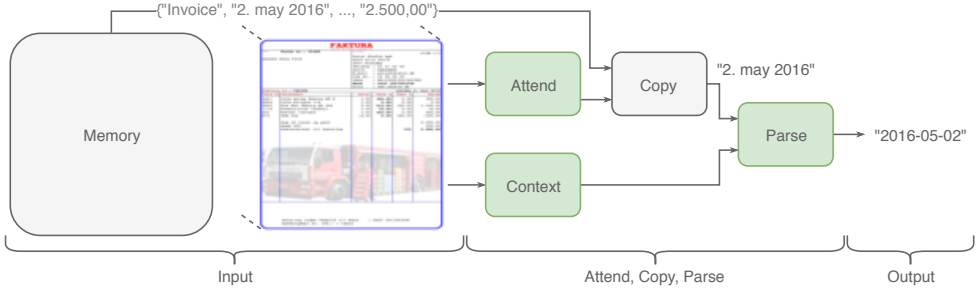


Figure 2.3: Overview of the Attend, Copy, Parse architecture. All modules are end-to-end differentiable. The modules highlighted in green are learned. From Palm et al. [2018a].

Second, the model can directly output, and be trained on, the end-to-end output, including normalized fields that require parsing, e.g. dates, amounts, etc. The architecture for accomplishing this is a bit involved, but is conceptually simple. For each field, the model extracts an N-gram from the input using an attention mechanism. This is similar to how the pointer network extracts a word in the Palm et al. [2017a]. The main difference is that the extracted N-gram is encoded as a sequence of characters. This sequence of characters is then passed through a learned neural parser, which parses it into the desired output format. The whole model is then trained end-to-end to produce the right sequence of characters. See figure 2.3 for an overview and table 2.3 for the results on new invoice layouts.

Field	Readable	Prod	Prod-	Attend, Copy, Parse
Number	0.90	0.78	0.78	**0.87**
Order id	0.90	0.82	0.82	**0.84**
Date	0.83	0.70	0.70	**0.80**
Total	0.81	**0.85**	0.77	0.81
Sub total	0.84	**0.84**	0.73	0.79
Tax total	0.80	**0.87**	0.77	0.80
Tax percent	0.79	0.83	0.68	**0.87**
Average	0.84	0.81	0.75	**0.83**

Table 2.3: Results. Fraction of correct values. Readable is the fraction of target values that can be found in the document using recall oriented parsers. "Prod" is a production system based on token classification and heuristics. "Prod-" is the "Prod" system, but with total heuristics disabled. From Palm et al. [2018a].

The proposed model can be seen as an end-to-end differentiable version of the system in Palm et al. [2017b]. In Palm et al. [2017b] the system is split into two parts: 1) the classifier which classifies the N-grams and 2) the heuristics which picks the most likely N-gram and parses it into the output. In the proposed model the attention over the N-grams can be seen as an implicit N-gram classifier, and the learned neural parsers directly replace the non-differentiable hand-crafted parsers. This is an instance of replacing a hard independence assumption with a soft structural prior. By learning the whole system end-to-end the model avoids the noise introduced when inferring the N-gram labels from the end-to-end data. Also, since the optimization objective now match the end-to-end measure of performance, better performance on the evaluation set directly translate into better end-to-end performance.

The proposed model performs better on four of the seven fields. The three fields it performs worse on are the total, sub total before tax, and tax total fields. The reason the baseline is better at these fields is because it uses a heuristic to select them jointly such that the totals add up. This heuristic significantly boosts the performance of the relatively weak baseline. In order to incorporate something like this, an end-to-end model needs to model the outputs that depend on each other jointly, which makes it an instance of a structured prediction problem. We discuss one idea for incorporating this heuristic in the paper, but it did not improve on the results.

The model stores pre-computed N-grams in the external memory. These N-grams are computed by a heuristic algorithm that assigns the words to lines and sorts them. In fact it's the same algorithm as used in Palm et al. [2017b]. This is unfortunate, since

this algorithm is not trained end-to-end and any errors it make cannot easily be solved by the rest of the model. If for instance it decides two words are on different lines because the document image is slightly rotated, it won't generate an N-gram of those two words, which means the attend module cannot attend to it. A better solution would be to store words in the external memory, and then recurrently attending to them, but this make the parsing much more difficult, and would be computationally more expensive.

2.3.4 Recurrent Relational Networks

***Rasmus Berg Palm**, Ulrich Paquet, Ole Winther* - Accepted at Conference on Neural Information Processing Systems (NIPS) 2018. See appendix D.

Palm et al. [2018b] introduces the Recurrent Relational Network (RRN). The RRN models dependent variables by operating on a graph of variables. The nodes in the graph represent the variables, represented by real valued vectors, and the edges represent the dependencies between the variables. The RRN is run for a pre-defined number of steps. At each step the variables "send" vector valued messages along the edges of the graph. The variables are then updated as a function of the incoming messages, their own states and their initial states. All the functions in the model are learned neural networks, and the whole model is trained in a supervised manner to output a fixed target for each variable at each step. We demonstrate the RRN on several datasets which require reasoning about dependent variables. Most notably the RRN solves 96.6% of the hardest Sudokus. This corresponds to finding the maximum (in fact, only) likely configuration of a joint distribution over 81 variables with complex dependencies. See figure 2.4 for an overview and table 2.4 for the results on the Sudoku dataset.

The RRN has three main advantages. 1) It's differentiable. As such it can be added to any neural network and trained end-to-end. Constraint propagation and search are powerful techniques, as shown in Norvig [2006], but they are not differentiable. The same is true for the total heuristic used by the baseline model in Palm et al. [2018a]. 2) It does not require you to specify how your variables depend on each other. The RRN that learns to solve Sudokus is never informed that digits in the same row, column and box must be unique. This is very useful in the case where you don't know how to formulate the dependencies. Indeed one of the early motivations for developing the RRN was the need to output the total fields that depended on each other, without knowing how to specify a proper joint model. 3) It does not require any special loss function or learning algorithm. It does not even need to be the last layer in the network. It is "just" a parameterized, differentiable function that operates on a graph of vectors, and output a graph of vectors.

The main limitation is that it ultimately models the dependent variables as condi-

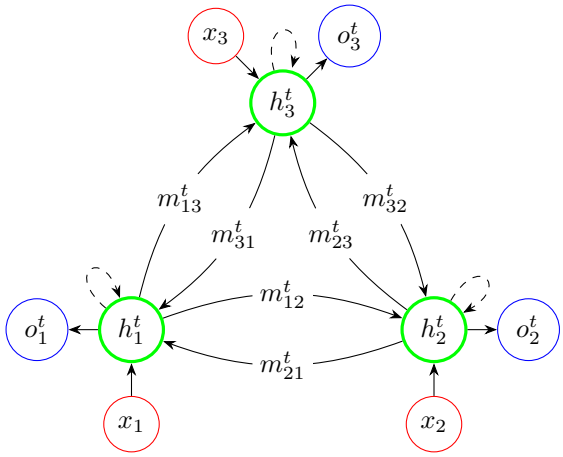


Figure 2.4: A *recurrent relational network* on a fully connected graph with 3 nodes. The nodes' hidden states h_i^t are highlighted with green, the inputs x_i with red, and the outputs o_i^t with blue. The dashed lines indicate the recurrent connections. Subscripts denote node indices and superscripts denote steps t. From Palm et al. [2018b].

tionally independent given the inputs. If the inputs $\mathbf{x}$, and outputs $\mathbf{y}$ both consists of N vectors s.t. $\mathbf{x} = [x_1, ..., x_N]$ and $\mathbf{y} = [y_1, ..., y_N]$, then the RRN model the joint probability as $p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|\mathbf{x})$. This is a false assumption of independence in many datasets, in which case $p(\mathbf{y}|\mathbf{x})$ cannot accurately model the true distribution. In practice it seems to work well though.

Method	Givens	Accuracy
*Recurrent Relational Network** (this work)	17	**96.6**%
Loopy BP, modified [Khan et al., 2014]	17	92.5%
Loopy BP, random [Bauke, 2008]	17	61.7%
Loopy BP, parallel [Bauke, 2008]	17	53.2%
Deeply Learned Messages* [Lin et al., 2015]	17	0%
Relational Network, node* [Santoro et al., 2017]	17	0%
Relational Network, graph* [Santoro et al., 2017]	17	0%
Deep Convolutional Network [Park, 2016]	24-36	70%

Table 2.4: Comparison of methods for solving Sudoku puzzles. Only methods that are differentiable are included in the comparison. Entries marked with an asterix are our own experiments, the rest are from the respective papers. From Palm et al. [2018b] .

CHAPTER 3

Conclusions

3.1 Conclusion

The goal of the project was to develop end-to-end deep learning models for extracting information from invoices. For simple fields, this has to a large degree been achieved. For the fields that depend on each other, e.g. the totals, there is still some way to go in successfully modelling the joint distribution. Outputting the lines present an even greater challenge, since these are both repeating and structured.

While the project have been very focused on invoices, the proposed models should be broadly applicable to end-to-end information extraction tasks. As discussed, information extraction tasks are ubiquitous, and end-to-end data is often the only available data. The presented research should be broadly applicable in these situations. While it's clear that there are great challenges remaining for end-to-end information extraction to be broadly successful, I hope the research presented in this thesis can serve as a starting point.

In particular I hope three contributions will be useful

1. The Attend, Copy, Parse architecture as a neural building block for end-to-end information extraction systems. Extracting and parsing snippets of text is arguable at the core of IE. This architecture shows how to do it in an end-to-end differentiable manner that can be used as part of a larger end-to-end IE architecture.

2. The Recurrent Relational Network, as a powerful neural module for relational reasoning and modelling dependent variables. Especially for cases where properly modelling the dependent variables are hard.

3. The input representation, where learned word embeddings are fused early with the document image. It's a general representation of document images that combine the text and image modalities in a natural way. While it is very similar to the approach taken in Yang et al. [2017], it is a very useful representation.

3.2 Future work

The interdependent total fields is an obvious candidate for future work. This is a challenging structured prediction problem with hard constraints on the variables. In this case there are four variables and two constraints: 1) `total = sub total + tax total` and 2) `tax total = sub total` $\times$ `tax percent`. In most cases these fields will not be verbatim present in the inputs so needs to be parsed before the constraints can even be evaluated. In some cases some of the fields will legitimately not be present in the input, e.g. if there's no tax, a zero tax total might not be present. In these cases the fields should instead be inferred from the constraints. One idea, is to extract and parse N candidates for each total field, then construct the N^4 combinations, and compute attention probabilities for each combination. Finally the output for each field would be a weighted sum over the field values in the combinations weighted by their attention probability. Another idea is to feed the $4N$ candidates into a RRN as a fully connected graph, and finally attend to a candidate for each of the fields.

Extracting the lines is an even harder task, with both individual and joint constraints. Individually each line should satisfy `line total = line price` $\times$ `line amount`, and jointly all the lines should satisfy `sub total` $= \sum$ `line totals`. All the same difficulties regarding the total fields apply. Additionally a variable number of lines must be extracted, potentially hundreds. Since many of the structured prediction difficulties are shared with the total fields, I think solving those first would be a sound approach.

At some point the errors introduced by the OCR engine might become the dominant source of errors. In this case looking at improving the OCR process might be relevant. I'd advice against training a new OCR engine from scratch as part of a fully end-to-end model. Fine tuning an existing OCR engine might be a better approach. Some OCR engines output n-best lists of word candidates or even individual character probability distributions. The latter would be especially interesting, since it fits naturally into the Attend, Copy, Parse architecture where the inputs to be parsed are encoded as a sequence of character distributions. This might allow the parser to correct commonly mistaken characters, e.g. 1 and 7.

The presented research has only considered the case of learning a single model of invoices that generalize to new suppliers. It would be interesting to see if one could obtain better results by learning individual models for each supplier, maybe initialized from a single global model.

APPENDIX A

CloudScan - A configuration-free invoice analysis system using recurrent neural networks.

Published at International Conference on Document Analysis and Recognition (ICDAR) 2017

CloudScan - A configuration-free invoice analysis system using recurrent neural networks

Rasmus Berg Palm
DTU Compute
Technical University of Denmark
rapal@dtu.dk

Ole Winther
DTU Compute
Technical University of Denmark
olwi@dtu.dk

Florian Laws
Tradeshift
Copenhagen, Denmark
fla@tradeshift.com

***Abstract*—We present CloudScan; an invoice analysis system that requires zero configuration or upfront annotation.**

In contrast to previous work, CloudScan does not rely on templates of invoice layout, instead it learns a single global model of invoices that naturally generalizes to unseen invoice layouts.

The model is trained using data automatically extracted from end-user provided feedback. This automatic training data extraction removes the requirement for users to annotate the data precisely.

We describe a recurrent neural network model that can capture long range context and compare it to a baseline logistic regression model corresponding to the current CloudScan production system.

We train and evaluate the system on 8 important fields using a dataset of 326,471 invoices. The recurrent neural network and baseline model achieve 0.891 and 0.887 average F1 scores respectively on seen invoice layouts. For the harder task of unseen invoice layouts, the recurrent neural network model outperforms the baseline with 0.840 average F1 compared to 0.788.

I. Introduction

Invoices, orders, credit notes and similar business documents carry the information needed for trade to occur between companies and much of it is on paper or in semi-structured formats such as PDFs [1]. In order to manage this information effectively, companies use IT systems to extract and digitize the relevant information contained in these documents. Traditionally this has been achieved using humans that manually extract the relevant information and input it into an IT system. This is a labor intensive and expensive process [2].

The field of information extraction addresses the challenge of automatically extracting such information and several commercial solutions exists that assist in this. Here we present CloudScan, a commercial solution by Tradeshift, free for small businesses, for extracting structured information from unstructured invoices.

Powerful information extraction techniques exists given that we can observe invoices from the same template beforehand, e.g. rule, keyword or layout based techniques. A template is a distinct invoice layout, typically unique to each sender. A number of systems have been proposed that rely on first classifying the template, e.g. Intellix [3], ITESOFT [4], smartFIX [5] and others [6], [7], [8]. As these systems rely on having seen the template beforehand, they cannot accurately handle documents from unseen templates. Instead they focus on requiring as few examples from a template as possible.

What is harder, and more useful, is a system that can accurately handle invoices from completely unseen templates, with no prior annotation, configuration or setup. This is the goal of CloudScan: to be a simple, configuration and maintenance free invoice analysis system that can convert documents from both previously seen and unseen templates with high levels of accuracy.

CloudScan was built from the ground up with this goal in mind. There is no notion of template in the system. Instead every invoice is processed by the same system built around a single machine learning model. CloudScan does not rely on any system integration or prior knowledge, e.g. databases of orders or customer names, meaning there is no setup required in order to use it.

CloudScan automatically extracts the training data from end-user provided feedback. The end-user provided feedback required is the correct value for each field, rather than the map from words on the page to fields. It is a subtle difference, but this separates the concerns of reviewing and correcting values using a graphical user interface from concerns related to acquiring training data. Automatically extracting the training data this way also results in a very large dataset which allows us to use methods that require such large datasets.

In this paper we describe how CloudScan works, and investigate how well it accomplishes the goal it aims to achieve. We evaluate CloudScan using a large dataset of 326,471 invoices and report competitive results on both seen and unseen templates. We establish two classification baselines using logistic regression and recurrent neural networks, respectively.

II. Related Work

The most directly related works are Intellix [3] by DocuWare and the work by ITESOFT [4]. Both systems require that relevant fields are annotated for a template manually beforehand, which creates a database of templates, fields and automatically extracted keywords and positions for each field. When new documents are received, both systems classify the template automatically using address lookups or machine learning classifiers. Once the template is classified the keywords and positions for each field are used to propose field candidates which are then scored using heuristics such as proximity and uniqueness of the keywords. Having scored the candidates the best one for each field is chosen.

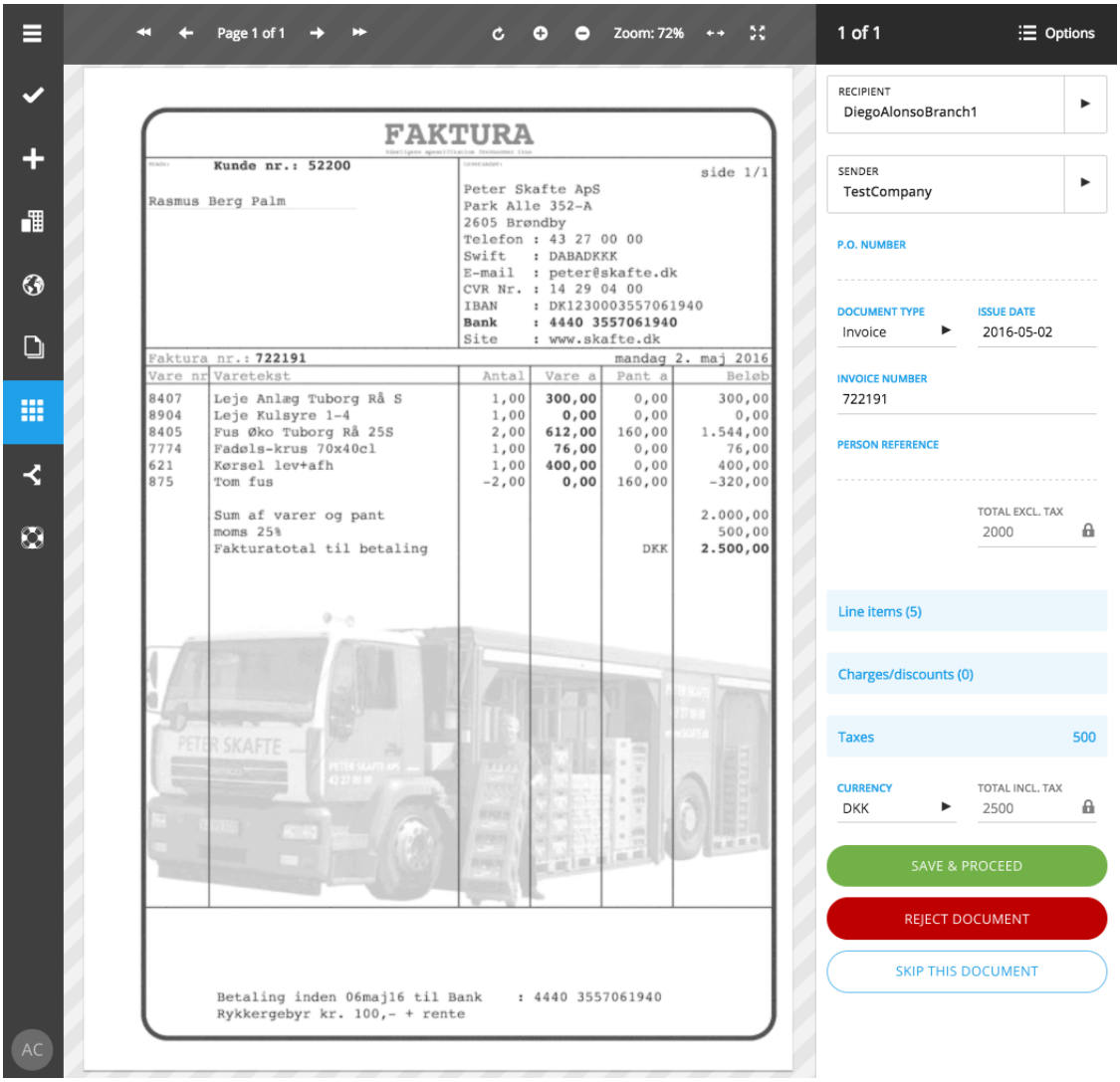


Fig. 1. The CloudScan graphical user interface. Results before any correction. Disregard the selected sender and recipient as these are limited to companies connected to the company uploading the invoice. This is an example of a perfect extraction which would give an F1 score of 1.

smartFIX [5] uses manually configured rules for each template. Cesarini et al. [6] learns a database of keywords for each template and fall back to a global database of keywords. Esser et al. [7] uses a database of absolute positions of fields for each template. Medvet et al. [8] uses a database of manually created (field, pattern, parser) triplets for each template, designs a probabilistic model for finding the most similar pattern in a template, and extracts the value with the associated parser.

Unfortunately we cannot compare ourselves directly to the works described as the datasets used are not publicly available and the evaluation methods are substantially different. However, the described systems all rely on having an annotated example from the same template in order to accurately extract information.

To the best of our knowledge CloudScan is the first invoice analysis system that is built for and capable of accurately converting invoices from unseen templates.

The previous works described can be configured to handle arbitrary document classes, not just invoices, as is the case for CloudScan. Additionally, they allow the user to define which

set of fields are to be extracted per class or template, whereas CloudScan assumes a single fixed set of fields to be extracted from all invoices.

Our automatic training data extraction is closely related to the idea of distant supervision [9] where relations are extracted from unstructured text automatically using heuristics.

The field of Natural Language Processing (NLP) offers a wealth of related work. Named Entity Recognition (NER) is the task of extracting named entities, usually persons or locations, from unstructured text. See Nadeau and Sekine [10] for a survey of NER approaches. Our system can be seen as a NER system in which we have 8 different entities. In recent years, neural architectures have been demonstrated to achieve state-of-the-art performance on NER tasks, e.g. Lample et al. [11], who combine word and character level RNNs, and Conditional Random Fields (CRFs).

Slot Filling is another related NLP task in which pre-defined slots must be filled from natural text. Our system can be seen as a slot filling task with 8 slots, and the text of a single invoice as input. Neural architectures are also used here, e.g. [12] uses bi-directional RNNs and word embedding to achieve competitive results on the ATIS (Airline Travel Information Systems) benchmark dataset.

In both NER and Slot Filling tasks, a commonly used approach is to classify individual tokens with the entities or slots of interest, an approach that we adopt in our proposed RNN model.

III. CloudScan

A. Overview

CloudScan is a cloud based software as a service invoice analysis system offered by Tradeshift. Users can upload their unstructured PDF invoices and the CloudScan engine converts them into structured XML invoices. The CloudScan engine contains 6 steps. See Figure 2.

1) **Text Extractor**. Input is a PDF invoice. Extracts words and their positions from the PDF. If the PDF has embedded text, the text is extracted, otherwise a commercial OCR engine is used. The output of this step is a structured representation of words and lines in hOCR format [13].
2) **N-grammer**. Creates N-grams of words on the same line. Output is a list of N-grams up to length 4.
3) **Feature Calculator**. Calculates features for every N-gram. Features fall in three categories: text, numeric and boolean. Examples of text features are the raw text of the N-gram, and the text after replacing all letters with "x", all numbers with "0" and all other characters with ".". Examples of numeric features are the normalized position on the page, the width and height and number of words to the left. Boolean features include whether the N-gram parses as a date or an amount or whether it matches a known country, city or zip code. These parsers and small databases of countries, cities and zip codes are built into the system, and does not require any configuration on the part of the user. The output is a feature vector for every N-gram. For a complete list of features see table V.
4) **Classifier**. Classifies each N-gram feature vector into 32 fields of interest, e.g. invoice number, total, date, etc. and one additional field 'undefined'. The undefined field is used for all N-grams that does not have a corresponding field in the output document, e.g. terms and conditions. The output is a vector of 33 probabilities for each N-gram.
5) **Post Processor**. Decides which N-grams are to be used for the fields in the output document. For all fields, we first filter out N-gram candidates that does not fit the syntax of the field after parsing with the associated parser. E.g. the N-gram "Foo Bar" would not fit the Total field after parsing with the associated parser since no amount could be extracted. The parsers can handle simple OCR errors and various formats, e.g. "100,0o" would be parsed to "100.00". The parsers are based on regular expressions.
   For fields with no semantic connection to other fields, e.g. the invoice number, date, etc. we use the Hungarian algorithm [14]. The Hungarian algorithm solves the assignment problem, in which N agents are to be assigned to M tasks, such that each task has exactly one agent assigned and no agent is assigned to more than one task. Given that each assignment has a cost, the Hungarian algorithm finds the assignments that minimizes the total cost. We use 1 minus the probability of an N-gram being a field as the cost.
   For the assignment of the Total, Line Total, Tax Total and Tax Percentage we define and minimize a cost function based on the field probabilities and whether the candidate totals adds up.
   The output is a mapping from the fields of interest to the chosen N-grams.
6) **Document Builder**. Builds a Universal Business Language (UBL) [15] invoice with the fields having the values of the found N-grams. UBL is a XML based invoice file format. Output is a UBL invoice.

B. Extracting training data from end-user provided feedback

The UBL invoice produced by the engine is presented to the user along with the original PDF invoice in a graphical user interface (GUI). The user can correct any field in the UBL invoice, either by copy and pasting from the PDF, or by directly typing in the correction. See figure 1.

Once the user has corrected any mistakes and accepted the invoice we add the resulting UBL to our data collection. We will extract training data from these validated UBL documents, even though they might deviate from the PDF content due to OCR error, user error or the user intentionally deviating from the PDF content. We discuss these issues later.

The classifier is trained on N-grams and their labels, which are automatically extracted from the validated UBL invoices and the corresponding PDFs. For each field in the validated

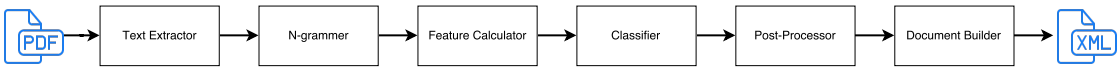


Fig. 2. The CloudScan engine.

UBL document we consider all N-grams in the PDF and check whether the text content, after parsing, matches the field value. If it does, we extract it as a single training example of N-gram and label equal to the field. If an N-gram does not match any fields we assign the 'undefined' label. For N-grams that match multiple fields, we assign all matched fields as labels. This ambiguity turns the multi-class problem into a multi-label problem. See Algorithm 1 for details.

```
input : UBL and PDF document
output: All labeled N-grams
result ← {};
foreach field ∈ fields do
    parser ← GetParser(field);
    value ← GetValue(UBL, field);
    maxN ← Length(value) + 2;
    nGrams ← CreateNgrams(PDF, maxN);
    foreach nGram ∈ nGrams do
        if value = Parse(nGram, parser) then
            Add(result, nGram, field);
        end
    end
end
nGrams ← CreateNgrams(PDF, 4);
foreach nGram ∈ nGrams do
    if nGram ∉ result then
        Add(result, nGram, undefined);
    end
end
return result
```

Algorithm 1: Automatic training data extraction

Using automatically extracted pairs like this results in a noisy, but big data set of millions of pairs. Most importantly, however, it introduces no limitations on how users correct potential errors, and requires no training. For instance, we could have required users to select the word matching a field, which would result in much higher quality training data. However in a high volume enterprise setup, this could reduce throughput significantly. Our automatic training data generation decouples the concerns of reviewing and correcting fields from creating training data, allowing the GUI to focus solely on reviewing and correcting fields. The user would need to review the field values and correct potential errors regardless, so as long as we do not limit how the user does it, we are not imposing any additional burdens. In short, the machine learning demands have lower priority than the user experience in this regard.

As long as we get a PDF and a corresponding UBL invoice we can extract training data, and the system should learn and improve for the next invoice.

IV. Experiments

We perform two experiments meant to test 1) the expected performance on the next invoice, and 2) the harder task of expected performance on the next invoice from an *unseen* template. These are two different measures of generalization performance.

The data set consists of 326,471 pairs of validated UBL invoices and corresponding PDFs from 8911 senders to 1013 receivers obtained from use of CloudScan. We assume each sender corresponds to a distinct template.

For the first experiment we split the invoices into a training, validation and test set randomly, using 70%, 10% and 20% respectively. For the second experiment we split the *senders* into a training, validation and test set randomly, using 70%, 10% and 20% respectively. All the invoices from the senders in a set then comprise the documents of that set. This split ensures that there are no invoices sharing templates between the three sets for the second experiment.

While the system captures 32 fields we only report on eight of them: Invoice number, Issue Date, Currency, Order ID, Total, Line Total, Tax Total and Tax Percent. We only report on these eight fields as they are the ones we have primarily designed the system for. A large part of the remaining fields are related to the sender and receiver of the invoice and used for identifying these. We plan to remove these fields entirely and approach the problem of sender and receiver identification as a document classification problem instead. Preliminary experiments based on a simple bag-of-words model show promising results. The last remaining fields are related to the line items and used for extracting these. Table extraction is a challenging research question in itself, and we are not yet ready to discuss our solution. Also, while not directly comparable, related work [3], [4], [6] also restricts evaluation to header fields.

Performance is measured by comparing the fields of the generated and validated UBL. Note we are not only measuring the classifier performance, but rather the performance of the entire system. The end-to-end performance is what is interesting to the user after all. Furthermore, this is the strictest possible way to measure performance, as it will penalize errors from any source, e.g. OCR errors and inconsistencies between the validated UBL and the PDF. For instance, the date in the validated UBL might not correspond to the date on the PDF. In this case, even if the date on the PDF is found, it will be counted as an error, as it does not match the date in the validated UBL.

In order to show the upper limit of the system under this measure we include a ceiling analysis where we replace the classifier output with the correct labels directly. This corresponds to using an oracle classifier. We use the MUC-5 definitions of recall, precision and F1, without partial matches [16].

We perform experiments with two classifiers 1) The produc-

tion baseline system using a logistic regression classifier, and 2) a Recurrent Neural Network (RNN) model. We hypothesize the RNN model can capture context better.

A. Baseline

The baseline is the current production system, which uses a logistic regression classifier to classify each N-gram individually.

In order to capture some context, we concatenate the feature vectors for the closest N-grams in the top, bottom, left and right directions to the normal feature vectors. So if the feature vector for an N-gram had M entries, after this it would have $5M$ entries.

All $5M$ features are then mapped to a binary vector of size 2^{22} using the hashing trick [17]. To be specific, for each feature we concatenate the feature name and value, hash it, take the remainder with respect to the binary vector size and set that index in the binary vector to 1.

The logistic regression classifier is trained using stochastic gradient descent for 10 epochs after which we see little improvement. This baseline system is derived from the heavily optimized winning solution of a competition Tradeshift held[1].

B. LSTM model

In order to accurately classify N-grams the context is critical, however when classifying each N-gram in isolation, as in the baseline model, we have to engineer features to capture this context, and deciding how much and which context to capture is not trivial.

A Recurrent Neural Network (RNN) can model the entire invoice and we hypothesize that this ability to take the entire invoice into account in a principled manner will improve the performance significantly. Further, it frees us from having to explicitly engineer features that capture context. As such we only use the original M features, not the $5M$ features of the baseline model. In general terms, a RNN can be described as follows.

$$h_t = f(h_{t-1}, x_t)$$
$$y_t = g(h_t)$$

Where h_t is the hidden state at step t, f is a neural network that maps the previous hidden state h_{t-1}, and the input x_t to h_t and g is a neural network that maps the hidden state h_t to the output of the model y_t. Several variants have been proposed, most notably the Long Short Term Memory (LSTM) [18] which is good at modeling long term dependencies.

A RNN models a sequence, i.e. x and y are ordered and as such we need to impose an ordering on the invoice. We chose to model the words instead of N-grams, as they fit the RNN sequence model more naturally and we use the standard left-to-right reading order as the ordering. Since the labels can span multiple words we re-label the words using the IOB labeling scheme [19]. The sequence of words "Total Amount: 12 200 USD" would be labeled "O O B-Total I-Total B-Currency".

We hash the text of the word into a binary vector of size 2^{18} which is embedded in a trainable 500 dimensional distributed representation using an embedding layer [20]. Using hashing instead of a fixed size dictionary is somewhat unorthodox but we did not observe any difference from using a dictionary, and hashing was easier to implement. It is possible we could have gotten better results using more advanced techniques like byte pair encoding [21].

We normalize the numerical and boolean features to have zero mean and unit variance and form the final feature vector for each word by concatenating the word embedding and the normalized numerical features.

From input to output, the model has: two dense layers with 600 rectified linear units each, a single bidirectional LSTM layer with 400 units, and two more dense layers with 600 rectified linear units each, and a final dense output layer with 65 logistic units (32 classes that can each be 'beginning' or 'inside' plus the 'outside' class).

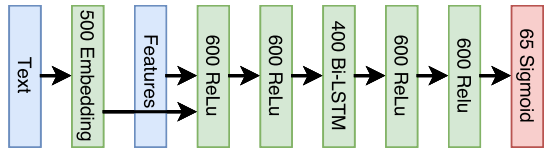


Fig. 3. The LSTM model.

Following Gal [22], we apply dropout on the recurrent units and on the word embedding using a dropout fraction of 0.5 for both. Without this dropout the model severely overfits.

The model is trained with the Adam optimizer [23] using minibatches of size 96 until the validation performance has not improved on the validation set for 5 epochs. Model architecture and hyper-parameters were chosen based on the performance on the validation set. For computational reasons we do not train on invoices with more than 1000 words, which constitutes approximately 5% of the training set, although we do test on them. The LSTM model was implemented in Theano [24] and Lasagne [25].

After classification we assign each word the IOB label with highest classification probability, and chunk the IOB labeled words back into labeled N-grams. During chunking, words with I labels without matching B labels are ignored. For example, the sequence of IOB labels [B-Currency, O, B-Total, I-Total, O, I-Total, O] would be chunked into [Currency, O, Total, O, O]. The labeled N-grams are used as input for the Post Processor and further processing is identical to the baseline system.

V. Results

The results of the ceiling analysis seen in Table I show that we can achieve very competitive results with CloudScan using an oracle classifier. This validates the overall system design,

[1]https://www.kaggle.com/c/tradeshift-text-classification

TABLE I
CEILING ANALYSIS RESULTS. MEASURED ON ALL DOCUMENTS. EXPECTED PERFORMANCE GIVEN AN ORACLE CLASSIFIER.

Field	F1	Precision	Recall
Number	0.918	0.967	0.873
Date	0.899	1.000	0.817
Currency	0.884	1.000	0.793
Order ID	0.820	0.979	0.706
Total	0.966	0.981	0.952
Line Total	0.976	0.991	0.961
Tax Total	0.959	0.961	0.957
Tax Percent	0.901	0.928	0.876
Micro avg.	0.925	0.974	0.881

including the use of automatically generated training data, and leaves us with the challenge of constructing a good classifier.

The attentive reader might wonder why the precision is not 1 exactly for all fields, when using the oracle classifier. For the 'Number' and 'Order ID' fields this is due to the automatic training data generation algorithm disregarding spaces when finding matching N-grams, whereas the comparison during evaluation is strict. For instance the automatic training data generator might generate the N-gram ("16 2054": Invoice Number) from (Invoice Number: "162054") in the validated UBL. When the oracle classifier classifies the N-gram "16 2054" as Invoice Number the produced UBL will be (Invoice Number: "16 2054"). When this is compared to the expected UBL of (Invoice Number: "162054") it is counted as incorrect. This is an annoying artifact of the evaluation method and training data generation. We could disregard spaces when comparing strings during evaluation, but we would risk regarding some actual errors as correct then. For the total fields and the tax percent, the post processor will attempt to calculate missing numbers from found numbers, which might result in errors.

As it stands the recall rate is the limiting factor of the system. The low recall rate can have two explanations: 1) The information is present in the PDF but we cannot read or parse it, e.g. it might be an OCR error or a strange date format, in which case the OCR engine or parsing should be improved, or 2) the information is legitimately not present in the PDF, in which case there is nothing to do, except change the validated UBL to match the PDF.

Table II shows the results of experiment 1 measuring the expected performance on the next received invoice for the baseline and LSTM model. The LSTM model is slightly better than the baseline system with an average F1 of 0.891 compared to 0.887. In general the performance of the models is very similar, and close to the theoretical maximum performance given by the ceiling analysis. This means the classifiers both perform close to optimally for this experiment. The gains that can be had from improving upon the LSTM model further are just 0.034 average F1.

More interesting are the results in Table III which measures

TABLE II
EXPECTED PERFORMANCE ON NEXT RECEIVED INVOICE. BEST RESULTS IN BOLD.

	F1		Precision		Recall	
Field	Baseline	LSTM	Baseline	LSTM	Baseline	LSTM
Number	**0.863**	0.860	**0.883**	0.877	**0.844**	0.843
Date	0.821	**0.828**	0.876	**0.893**	**0.773**	0.772
Currency	0.869	**0.874**	0.974	**0.992**	**0.784**	0.781
Order ID	**0.776**	0.760	**0.936**	0.930	**0.663**	0.642
Total	0.927	**0.932**	0.940	**0.942**	0.915	**0.924**
Line Total	0.923	**0.936**	0.936	**0.945**	0.911	**0.927**
Tax Total	0.931	**0.939**	0.933	**0.941**	0.929	**0.937**
Tax Percent	0.901	**0.903**	0.927	**0.930**	0.876	**0.878**
Micro avg.	0.887	**0.891**	0.924	**0.930**	0.852	**0.855**

TABLE III
EXPECTED PERFORMANCE ON NEXT INVOICE FROM UNSEEN TEMPLATE. BEST RESULTS IN BOLD.

	F1		Precision		Recall	
Field	Baseline	LSTM	Baseline	LSTM	Baseline	LSTM
Number	0.711	**0.760**	0.761	**0.789**	0.668	**0.733**
Date	0.693	**0.774**	0.759	**0.847**	0.637	**0.712**
Currency	**0.907**	0.905	0.977	**0.983**	**0.847**	0.838
Order ID	0.433	**0.523**	**0.822**	0.737	0.294	**0.406**
Total	0.840	**0.896**	0.864	**0.907**	0.818	**0.884**
Line Total	0.803	**0.880**	0.826	**0.891**	0.781	**0.869**
Tax Total	0.832	**0.878**	0.835	**0.881**	0.829	**0.874**
Tax Percent	0.812	**0.869**	0.828	**0.887**	0.796	**0.853**
Micro avg.	0.788	**0.840**	0.836	**0.879**	0.746	**0.804**

the expected performance on the next invoice from an unseen template. This measures the generalization performance of the system across templates which is a much harder task due to the plurality of invoice layouts and reflects the experience a new user will have the first time they use the system. On this harder task the LSTM model clearly outperform the baseline system with an average F1 of 0.840 compared to 0.788. Notably the 0.840 average F1 of the LSTM model is getting close to the 0.891 average F1 of experiment 1, indicating that the LSTM model is largely learning a template invariant model of invoices, i.e. it is picking up on general patterns rather than just memorizing specific templates.

We hypothesized that it is the ability of LSTMs to model context directly that leads to increased performance, although there are several other possibilities given the differences between the two models. For instance, it could simply be that the LSTM model has more parameters, the non-linear feature combinations, or the word embedding.

To test our hypothesis we trained a third model that is identical to the LSTM model, except that the bidirectional LSTM layer was replaced with a feedforward layer with an equivalent number of parameters. We trained the network

with and without dropout, with all other hyper parameters kept equal. The best model got an average F1 of 0.702 on the experiment 2 split, which is markedly worse than both the LSTM and baseline model. Given that the only difference between this model and the LSTM model is the lack of recurrent connections we feel fairly confident that our hypothesis is true. The feedforward model is likely worse than the baseline model because it does not have the additional context features of the baseline model.

TABLE IV
WORD EMBEDDING EXAMPLES.

EUR	GBP	DKK
$	USD	DKK
Total	Betrag	TOTAL
Number	No	number
Number:	No	Rechnung-Nr.
London	LONDON	Bremen
Brutto	Ldm	ex.Vat
Phone:	code:	Tel:

Table IV shows examples of words and the two closest words in the learned word embedding. It shows that the learned embeddings are language agnostic, e.g. the closest word to "Total" is "Betrag" which is German for "Sum" or "Amount". The embedding also captures common abbreviations, capitalization, currency symbols and even semantic similarities such as cities. Learning these similarities versus encoding them by hand is a major advantage as it happens automatically as it is needed. If a new abbreviation, language, currency, etc. is encountered it will automatically be learned.

VI. DISCUSSION

We have presented our goals for CloudScan and described how it works. We hypothesized that the ability of a LSTM to model context directly would improve performance. We carried out experiments to test our hypothesis and evaluated CloudScan's performance on a large realistic dataset. We validated our hypothesis and showed competitive results of 0.891 average F1 on documents from seen templates, and 0.840 on documents from unseen templates using a single LSTM model. These numbers should be compared to a ceiling of F1=0.925 for an ideal system baseline where an oracle classifier is used.

Unfortunately it is hard to compare to other vendors directly as no large publicly available datasets exists due to the sensitive nature of invoices. We sincerely wish such a dataset existed and believe it would drive the field forward significantly, as seen in other fields, e.g. the large effect ImageNet [26] had on the computer vision field. Unfortunately we are not able to release our own dataset due to privacy restrictions.

A drawback of the LSTM model is that we have to decide upon an ordering of the words, when there is none naturally. We chose the left to right reading order which worked well, but in line with the general theme of CloudScan we would prefer a model which could learn this ordering or did not require one.

CloudScan works only on the word level, meaning it does not take any image features into account, e.g. the lines, logos, background, etc. We could likely improve the performance if we included these image features in the model.

With the improved results from the LSTM model we are getting close to the theoretical maximum given by the ceiling analysis. For unseen templates we can at maximum improve the average F1 by 0.085 by improving the classifier. This corresponds roughly to the 0.075 average F1 that can at maximum be gained from fixing the errors made under the ceiling analysis. An informal review of the errors made by the system under the ceiling analysis indicates the greatest source of errors are OCR errors and discrepancies between the validated UBL and the PDF.

As such, in order to substantially improve CloudScan we believe a two pronged strategy is required: 1) improve the classifier and 2) correct discrepancies between the validated UBL and PDF. Importantly, the second does not delay the turnaround time for the users, can be done at our own pace and only needs to be done for the cases where the automatic training data generation fails. As for the OCR errors we will rely on further advances in OCR technology.

ACKNOWLEDGMENT

We would like to thank Ángel Diego Cuñado Alonso and Johannes Ulén for our fruitful discussions, and their great work on CloudScan. This research was supported by the NVIDIA Corporation with the donation of TITAN X GPUs. This work is partly funded by the Innovation Fund Denmark (IFD) under File No. 5016-00101B.

REFERENCES

[1] A. J. Sellen and R. H. Harper, *The Myth of the Paperless Office*. Cambridge, MA, USA: MIT Press, 2003.

[2] B. Klein, S. Agne, and A. Dengel, "Results of a Study on Invoice-Reading Systems in Germany," in *Document Analysis Systems VI*, ser. Lecture Notes in Computer Science, S. Marinai and A. R. Dengel, Eds. Springer Berlin Heidelberg, Sep. 2004, no. 3163, pp. 451–462.

[3] D. Schuster, K. Muthmann, D. Esser, A. Schill, M. Berger, C. Weidling, K. Aliyev, and A. Hofmeier, "Intellix – End-User Trained Information Extraction for Document Archiving," in *2013 12th International Conference on Document Analysis and Recognition*, Aug. 2013, pp. 101–105.

[4] M. Rusiñol, T. Benkhelfallah, and V. P. dAndecy, "Field Extraction from Administrative Documents by Incremental Structural Templates," in *2013 12th International Conference on Document Analysis and Recognition*, Aug. 2013, pp. 1100–1104.

[5] A. Dengel and B. Klein, "smartFIX: A Requirements-Driven System for Document Analysis and Understanding," in *Proceedings of the 5th International Workshop on Document Analysis Systems V*, ser. DAS '02. London, UK, UK: Springer-Verlag, 2002, pp. 433–444.

[6] F. Cesarini, E. Francesconi, M. Gori, and G. Soda, "Analysis and understanding of multi-class invoices," *Document Analysis and Recognition*, vol. 6, no. 2, pp. 102–114, Oct. 2003. [Online]. Available: http://link.springer.com/article/10.1007/s10032-002-0084-6

[7] D. Esser, D. Schuster, K. Muthmann, M. Berger, and A. Schill, "Automatic Indexing of Scanned Documents - a Layout-based Approach," *Document Recognition and Retrieval XIX (DRR), San Francisco, CA, USA*, 2012. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1284003

[8] E. Medvet, A. Bartoli, and G. Davanzo, "A probabilistic approach to printed document understanding," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 14, no. 4, pp. 335–347, Nov. 2010. [Online]. Available: http://link.springer.com/article/10.1007/s10032-010-0137-1

[9] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, 2009, pp. 1003–1011.

[10] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[11] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural Architectures for Named Entity Recognition," 2016.

[12] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding." in *INTERSPEECH*, 2013, pp. 3771–3775.

[13] T. Breuel, "The hOCR Microformat for OCR Workflow and Results," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2, Sep. 2007, pp. 1063–1067.

[14] H. W. Kuhn, "The Hungarian method for the assignment problem," *Naval Research Logistics Quarterly*, vol. 2, no. 1-2, pp. 83–97, Mar. 1955. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/nav.3800020109/abstract

[15] G. K. Holman, "Universal business language v2.0," 2006.

[16] N. Chinchor and B. Sundheim, "MUC-5 Evaluation Metrics," in *Proceedings of the 5th Conference on Message Understanding*, ser. MUC5 '93. Association for Computational Linguistics, 1993, pp. 69–78.

[17] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature Hashing for Large Scale Multitask Learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: ACM, 2009, pp. 1113–1120. [Online]. Available: http://doi.acm.org/10.1145/1553374.1553516

[18] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[19] L. A. Ramshaw and M. P. Marcus, "Text Chunking Using Transformation-Based Learning," *Proceedings of the Third ACL Workshop on Very Large Corpora*, pp. 82–94, 1995.

[20] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A Neural Probabilistic Language Model," *Journal of Machine Learning Research*, vol. 3, no. Feb, pp. 1137–1155, 2003. [Online]. Available: http://www.jmlr.org/papers/v3/bengio03a.html

[21] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[22] Y. Gal, "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks," *arXiv:1512.05287 [stat]*, Dec. 2015, arXiv: 1512.05287. [Online]. Available: http://arxiv.org/abs/1512.05287

[23] D. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Dec. 2014, arXiv: 1412.6980. [Online]. Available: http://arxiv.org/abs/1412.6980

[24] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[25] S. Dieleman, J. Schlüter, C. Raffel, E. Olson, S. K. Sønderby, D. Nouri, D. Maturana, M. Thoma, E. Battenberg, J. Kelly, J. D. Fauw, M. Heilman, D. M. d. Almeida, B. McFee, H. Weideman, G. Takács, P. d. Rivaz, J. Crall, G. Sanders, K. Rasul, C. Liu, G. French, and J. Degrave, "Lasagne: First release." Aug. 2015. [Online]. Available: http://dx.doi.org/10.5281/zenodo.27878

[26] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

TABLE V
N-GRAM FEATURES.

Name	Description
RawText	The raw text.
RawTextLastWord	The raw text of the last word in the N-gram.
TextOfTwoWordsLeft	The raw text of the word two places to the left of the N-gram.
TextPatterns	The raw text, after replacing uppercase characters with X, lowercase with x, numbers with 0, repeating whitespace with single whitespace and the rest with ?.
bottomMargin	Vertical coordinate of the bottom margin of the N-gram normalized to the page height.
topMargin	Same as above, but for the top margin.
rightMargin	Horizontal coordinate of the right margin of the N-gram normalized to the page width.
leftMargin	Same as above but for the left margin.
bottomMarginRelative	The vertical distance to the nearest word below this N-gram, normalized to page height.
topMarginRelative	The vertical distance to the nearest word above this N-gram, normalized to page height.
rightMarginRelative	The horizontal distance to the nearest word to the right of this N-gram, normalized to page width.
leftMarginRelative	The horizontal distance to the nearest word to the left of this N-gram, normalized to page width.
horizontalPosition	The horizontal distance between this N-gram and the word to the left, normalized to the horizontal distance between the word to the left and the word to the right.
verticalPosition	Same as above but vertical.
hasDigits	Whether there are any digits 0-9 in the N-gram.
isKnownCity	Whether the N-gram is found in a small database of known cities.
isKnownCountry	Same as above, but for countries.
isKnownZip	Same as above but for zip codes.
leftAlignment	Number of words on the same page which left margin is within 5 pixels of this N-grams left margin.
length	Number of characters in the N-gram.
pageHeight	The height of the page of this N-gram.
pageWidth	The width of the page of this N-gram.
positionOnLine	Count of words to the left of this N-gram normalized to the count of total words on this line
lineSize	The number of words on this line.
lineWhiteSpace	The area occupied by whitespace on the line of this N-gram normalized to the total area of the line.
parsesAsAmount	Whether the N-gram parses as a fractional amount.
parsesAsDate	Same as above but for dates.
parsesAsNumber	Same as above but for integers.
LineMathFeatures.isFactor	Whether this N-gram, after parsing, can take part in an equation such that it is one of two factors on the same line that when multiplied equals another amount on the same line.
LineMathFeatures.isProduct	Same as above, except this N-gram is the product of the two factors.

APPENDIX B

End-to-End Information Extraction without Token-Level Supervision

Published at Workshop on Speech-Centric Natural Language Processing (SCNLP) at Conference on Empirical Methods in Natural Language Processing (EMNLP) 2017

End-to-End Information Extraction without Token-Level Supervision

Rasmus Berg Palm
DTU Compute
Technical University of Denmark
rapal@dtu.dk

Dirk Hovy
Computer Science Dpeartment
University of Copenhagen
dirk.hovy@di.ku.dk

Florian Laws
Tradeshift
Landemærket 10, 1119 Copenhagen
fla@tradeshift.com

Ole Winther
DTU Compute
Technical University of Denmark
olwi@dtu.dk

Abstract

Most state-of-the-art information extraction approaches rely on token-level labels to find the areas of interest in text. Unfortunately, these labels are time-consuming and costly to create, and consequently, not available for many real-life IE tasks. To make matters worse, token-level labels are usually not the desired output, but just an intermediary step. End-to-end (E2E) models, which take raw text as input and produce the desired output directly, need not depend on token-level labels. We propose an E2E model based on pointer networks, which can be trained directly on pairs of raw input and output text. We evaluate our model on the ATIS data set, MIT restaurant corpus and the MIT movie corpus and compare to neural baselines that do use token-level labels. We achieve competitive results, within a few percentage points of the baselines, showing the feasibility of E2E information extraction without the need for token-level labels. This opens up new possibilities, as for many tasks currently addressed by human extractors, raw input and output data are available, but not token-level labels.

1 Introduction

Humans spend countless hours extracting structured machine readable information from unstructured information in a multitude of domains. Promising to automate this, information extraction (IE) is one of the most sought-after industrial applications of natural language processing. However, despite substantial research efforts, in practice, many applications still rely on manual effort to extract the relevant information.

One of the main bottlenecks is a shortage of the data required to train state-of-the-art IE models, which rely on sequence tagging (Finkel et al., 2005; Zhai et al., 2017). Such models require sufficient amounts of training data that is labeled at the token-level, i.e., with one label for each word.

The reason token-level labels are in short supply is that they are not the intended output of human IE tasks. Creating token-level labels thus requires an additional effort, essentially doubling the work required to process each item. This additional effort is expensive and infeasible for many production systems: estimates put the average cost for a sentence at about 3 dollars, and about half an hour annotator time (Alonso et al., 2016). Consequently, state-of-the-art IE approaches, relying on sequence taggers, cannot be applied to many real life IE tasks.

What is readily available in abundance and at no additional costs, is the raw, unstructured input and machine-readable output to a human IE task. Consider the transcription of receipts, checks, or business documents, where the input is an unstructured PDF and the output a row in a database (due date, payable amount, etc). Another example is flight bookings, where the input is a natural language request from the user, and the output a HTTP request, sent to the airline booking API.

To better exploit such existing data sources, we propose an end-to-end (E2E) model based on pointer networks with attention, which can be trained end-to-end on the input/output pairs of human IE tasks, without requiring token-level annotations.

We evaluate our model on three traditional IE data sets. Note that our model and the baselines are competing in two dimensions. The first is cost and applicability. The baselines require token-level labels, which are expensive and unavailable for many real life tasks. Our model does *not* re-

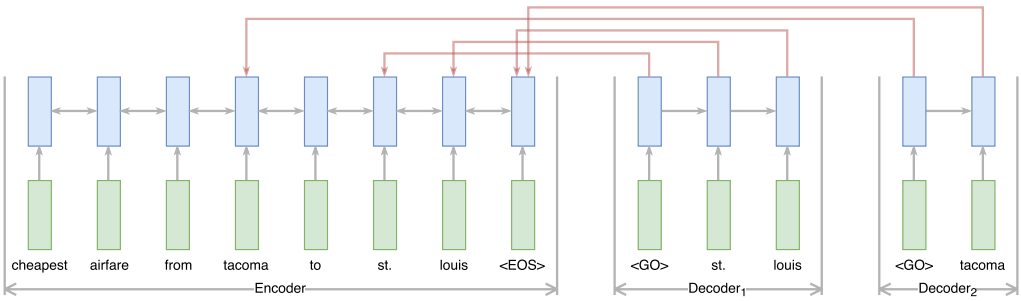


Figure 1: Our model based on pointer networks. The solid red lines are the attention weights. For clarity only two decoders are drawn and only the strongest attention weight for each output is drawn.

quire such token-level labels. Given the time and money required for these annotations, our model clearly improves over the baselines in this dimension. The second dimension is the accuracy of the models. Here we show that our model is competitive with the baseline models on two of the data sets and only slightly worse on the last data set, all despite fewer available annotations.

Contributions We present an E2E IE model with attention that does not depend on costly token-level labels, yet performs competitively with neural baseline models that rely on token-level labels. By saving both time and money at comparable performance, our model presents a viable alternative for many real-life IE needs. Code is available at github.com/rasmusbergpalm/e2e-ie-release

2 Model

Our proposed model is based on pointer networks (Vinyals et al., 2015). A pointer network is a sequence-to-sequence model with attention in which the output is a position in the input sequence. The input position is "pointed to" using the attention mechanism. See figure 1 for an overview. Our formulation of the pointer network is slightly different from the original: Our output is some content from the input rather than a position in the input.

An input sequence of N words $\mathbf{x} = x_1, ..., x_N$ is encoded into another sequence of length N using an Encoder.

$$e_i = \text{Encoder}(x_i, e_{i-1}) \tag{1}$$

We use a single shared encoder, and $k = 1..K$ decoders, one for each piece of information we wish to extract. At each step j each decoder calculate an unnormalized scalar attention score a_{kji} over each input position i. The k'th decoder output at step j, o_{kj}, is then the weighted sum of inputs, weighted with the normalized attention scores att_{kji}.

$$d_{kj} = \text{Decoder}_k(o_{k,j-1}, d_{k,j-1}) \tag{2}$$

$$a_{kji} = \text{Attention}_k(d_{kj}, e_i) \text{ for } i = 1..N \tag{3}$$

$$att_{kji} = \text{softmax}(a_{kji}) \text{ for } i = 1..N \tag{4}$$

$$o_{kj} = \sum_{i=1}^{N} att_{kji}\, x_i \ . \tag{5}$$

Since each x_i is a one-hot encoded word, and the att_{kji} sum to one over i, o_{kj} is a probability distribution over words.

The loss function is the sum of the negative cross entropy for each of the expected outputs y_{kj} and decoder outputs o_{kj}.

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\sum_{k=1}^{K} \frac{1}{M_k} \sum_{j=1}^{M_k} y_{kj} \log\left(o_{kj}\right) \ , \tag{6}$$

where M_k is the sequence length of expected output y_k.

The specific architecture depends on the choice of Encoder, Decoder and Attention. For the encoder, we use a Bi-LSTM with 128 hidden units and a word embedding of 96 dimensions. We use a separate decoder for each of the fields. Each decoder has a word embedding of 96 dimensions, a LSTM with 128 units, with a learned first hidden state and its own attention mechanism. Our attention mechanism follows Bahdanau et al. (2014)

$$a_{ji} = v^T \tanh(W_e\, enc_i + W_d\, dec_j) \ . \tag{7}$$

The attention parameters W_e, W_d and v for each attention mechanism are all 128-dimensional.

During training we use teacher forcing for the decoders (Williams and Zipser, 1989), such that $o_{k,j-1} = y_{k,j-1}$. During testing we use argmax to select the most probable output for each step j and run each decoder until the first end of sentence (EOS) symbol.

3 Experiments

3.1 Data sets

To compare our model to baselines relying on token-level labels we use existing data sets for which token level-labels are available. We measure our performance on the ATIS data set (Price, 1990) (4978 training samples, 893 testing samples) and the MIT restaurant (7660 train, 1521 test) and movie corpus (9775 train, 2443 test) (Liu et al., 2013). These data sets contains token-level labels in the Beginning-Inside-Out format (BIO).

The ATIS data set consists of natural language requests to a simulated airline booking system. Each word is labeled with one of several classes, e.g. departure city, arrival city, cost, etc. The MIT restaurant and movie corpus are similar, except for a restaurant and movie domain respectively. See table 1 for samples.

MIT Restaurant		**MIT Movie**	
2	B-Rating	show	O
start	I-Rating	me	O
restaurants	O	films	O
with	O	elvis	B-ACTOR
inside	B-Amenity	films	O
dining	I-Amenity	set	B-PLOT
		in	I-PLOT
		hawaii	I-PLOT

Table 1: Samples from the MIT restaurant and movie corpus. The transcription errors are part of the data.

Since our model does not need token-level labels, we create an E2E version of each data set without token-level labels by chunking the BIO-labeled words and using the labels as fields to extract. If there are multiple outputs for a single field, e.g. multiple destination cities, we join them with a comma. For the ATIS data set, we choose the 10 most common labels, and we use all the labels for the movie and restaurant corpus. The movie data set has 12 fields and the restaurant has 8. See Table 2 for an example of the E2E ATIS data set.

Input
cheapest airfare from tacoma to st. louis and detroit

Output	
fromloc	tacoma
toloc	st. louis , detroit
airline_name	-
cost_relative	cheapest
period_of_day	-
time	-
time_relative	-
day_name	-
day_number	-
month_name	-

Table 2: Sample from the E2E ATIS data set.

3.2 Baselines

For the baselines, we use a two layer neural network model. The first layer is a Bi-directional Long Short Term Memory network (Hochreiter and Schmidhuber, 1997) (Bi-LSTM) and the second layer is a forward-only LSTM. Both layers have 128 hidden units. We use a trained word embedding of size 128. The baseline is trained with Adam (Kingma and Ba, 2014) on the BIO labels and uses early stopping on a held out validation set.

This baseline architecture achieves a fairly strong F1 score of 0.9456 on the ATIS data set. For comparison, the published state-of-the-art is at 0.9586 (Zhai et al., 2017). These numbers are for the traditional BIO token level measure of performance using the publicly available conlleval script. They should not be confused with the E2E performance reported later. We present them here so that readers familiar with the ATIS data set can evaluate the strength of our baselines using a well-known measure.

For the E2E performance measure we train the baseline models using token-level BIO labels and predict BIO labels on the test set. Given the predicted BIO labels, we create the E2E output for the baseline models in the same way we created the E2E data sets, i.e. by chunking and extracting labels as fields. We evaluate our model and the baselines using the MUC-5 definitions of precision, recall and F1, without partial matches (Chinchor and

Sundheim, 1993). We use bootstrap sampling to estimate the probability that the model with the best micro average F1 score on the entire test set is worse for a randomly sampled subset of the test data.

3.3 Our model

Since our decoders can only output values that are present in the input, we prepend a single comma to every input sequence. We optimize our model using Adam and use early stopping on a held-out validation set. The model quickly converges to optimal performance, usually after around 5000 updates after which it starts overfitting.

For the restaurant data set, to increase performance, we double the sizes of all the parameters and use embedding and recurrent dropout following (Gal, 2015). Further, we add a summarizer LSTM to each decoder. The summarizer LSTM reads the entire encoded input. The last hidden state of the summarizer LSTM is then concatenated to each input to the decoder.

3.4 Results

Data set	Baseline	Ours	p
ATIS	0.977	0.974	0.1755
Movie	0.816	0.817	0.3792
Restaurant	**0.724**	0.694	0.0001

Table 3: Micro average F1 scores on the E2E data sets. Results that are significantly better ($p < 0.05$) are highlighted in bold.

We see in Table 3 that our model is competitive with the baseline models in terms of micro-averaged F1 for two of the three data sets. This is a remarkable result given that the baselines are trained on token-level labels, whereas our model is trained end-to-end. For the restaurant data set, our model is slightly worse than the baseline.

4 Related work

Event extraction (EE) is similar to the E2E IE task we propose, except that it can have several event types and multiple events per input. In our E2E IE task, we only have a single event type and assume there is zero or one event mentioned in the input, which is an easier task. Recently, Nguyen et al. (2016) achieved state of the art results on the ACE 2005 EE data set using a recurrent neural network to jointly model event triggers and argument roles.

Other approaches have addressed the need for token-level labels when only raw output values are available. Mintz et al. (2009) introduced distant supervision, which heuristically generates the token-level labels from the output values. You do this by searching for input tokens that matches output values. The matching tokens are then assigned the labels for the matching outputs. One drawback is that the quality of the labels crucially depend on the search algorithm and how closely the tokens match the output values, which makes it brittle. Our method is trained end-to-end, thus not relying on brittle heuristics.

Sutskever et al. (2014) opened up the sequence-to-sequence paradigm. With the addition of attention (Bahdanau et al., 2014), these models achieved state-of-the-art results in machine translation (Wu et al., 2016). We are broadly inspired by these results to investigate E2E models for IE.

The idea of copying words from the input to the output have been used in machine translation to overcome problems with out-of-vocabulary words (Gulcehre et al., 2016; Gu et al., 2016).

5 Discussion

We present an end-to-end IE model that does not require detailed token-level labels. Despite being trained end-to-end, it is competitive with baseline models relying on token-level labels. In contrast to them, our model can be used on many real life IE tasks where intermediate token-level labels are not available and creating them is not feasible.

In our experiments our model and the baselines had access to the same amount of training samples. In a real life scenario it is likely that token-level labels only exist for a subset of all the data. It would be interesting to investigate the quantity/quality trade-of of the labels, and a multi task extension to the model, which could make use of available token-level labels.

Our model is remarkably stable to hyper parameter changes. On the restaurant dataset we tried several different architectures and hyper parameters before settling on the reported one. The difference between the worst and the best was approximately 2 percentage points.

A major limitation of the proposed model is that it can only output values that are present in the input. This is a problem for outputs that are normalized before being submitted as machine readable data, which is a common occurrence. For instance, dates might appear as '`Jan 17 2012`' in

the input and as '17-01-2012' in the machine readable output.

While it is clear that this model does not solve all the problems present in real-life IE tasks, we believe it is an important step towards applicable E2E IE systems.

In the future, we will experiment with adding character level models on top of the pointer network outputs so the model can focus on an input, and then normalize it to fit the normalized outputs.

Acknowledgments

We would like to thank the reviewers who helped make the paper more concise. Dirk Hovy was supported by the Eurostars grant E10138 ReProsis. This research was supported by the NVIDIA Corporation with the donation of TITAN X GPUs.

References

Héctor Martínez Alonso, Djamé Seddah, and Benoît Sagot. 2016. From Noisy Questions to Minecraft Texts: Annotation Challenges in Extreme Syntax Scenarios. *WNUT 2016* page 13.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

Nancy Chinchor and Beth Sundheim. 1993. MUC-5 Evaluation Metrics. In *Proceedings of the 5th Conference on Message Understanding*. Association for Computational Linguistics, MUC5 '93, pages 69–78. https://doi.org/10.3115/1072017.1072026.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, Stroudsburg, PA, USA, ACL '05, pages 363–370.

Yarin Gal. 2015. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *arXiv:1512.05287 [stat]* ArXiv: 1512.05287. http://arxiv.org/abs/1512.05287.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor OK Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. *arXiv preprint arXiv:1603.06393* .

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. *arXiv preprint arXiv:1603.08148* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9(8):1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* ArXiv: 1412.6980. http://arxiv.org/abs/1412.6980.

Jingjing Liu, Panupong Pasupat, Scott Cyphers, and Jim Glass. 2013. Asgard: A portable architecture for multilingual dialogue systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, pages 8386–8390.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. 2009. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics, pages 1003–1011.

Thien Huu Nguyen, Kyunghyun Cho, and Ralph Grishman. 2016. Joint event extraction via recurrent neural networks. In *Proceedings of NAACL-HLT*. pages 300–309.

Patti Price. 1990. Evaluation of spoken language systems: The ATIS domain. In *Proceedings of the Third DARPA Speech and Natural Language Workshop*. Morgan Kaufmann, pages 91–95.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*. pages 3104–3112.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*. pages 2692–2700.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1(2):270–280.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, and others. 2016. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *arXiv preprint arXiv:1609.08144* .

Feifei Zhai, Saloni Potdar, Bing Xiang, and Bowen Zhou. 2017. Neural Models for Sequence Chunking. *arXiv preprint arXiv:1701.04027* .

APPENDIX C

Attend, Copy, Parse - End-to-end information extraction from documents

Unpublished

Attend, Copy, Parse
End-to-end information extraction from documents

Rasmus Berg Palm
DTU Compute
Tradeshift
rapal@dtu.dk

Florian Laws
Tradeshift
fla@tradeshift.com

Ole Winther
DTU Compute
olwi@dtu.dk

***Abstract*—Document information extraction tasks performed by humans create data consisting of a PDF or document image input, and extracted string outputs. This end-to-end data is naturally consumed and produced when performing the task because it is valuable *in and of itself*. It is naturally available, at no additional cost. Unfortunately, state-of-the-art word classification methods for information extraction cannot use this data, instead requiring word-level labels which are expensive to create and consequently not available for many real life tasks. In this paper we propose the Attend, Copy, Parse architecture, a deep neural network model that can be trained directly on end-to-end data, bypassing the need for word-level labels. We evaluate the proposed architecture on a large diverse set of invoices, and outperform a state-of-the-art production system based on word classification. We believe our proposed architecture can be used on many real life information extraction tasks where word classification cannot be used due to a lack of the required word-level labels.**

I. Introduction

As long as people communicate using unstructured documents, there'll be a demand for extracting structured information from these documents. However, extracting information from such documents is a tedious and costly task for humans. The field of information extraction investigates how to automate this task.

Consider employees at an enterprise processing invoices. They receive a paper or PDF invoice, extract a few important fields, e.g. the invoice number, total, due date, etc, and type it into a computer system. Simply doing their job they are implicitly creating a dataset consisting of pairs of PDF or paper invoices and their extracted fields. We define end-to-end data as such data that is naturally consumed and produced in a human information extraction tasks. By definition this data is available, should one wish to capture it.

Unfortunately such end-to-end data cannot be used with state-of-the-art machine learning methods for information extraction. Current state-of-the-art approaches require labeling of every word, which is costly to obtain, and consequently not available for many real life tasks. The distinction between end-to-end data and data labeled on the word level is subtle but important. In the invoice example the end-to-end data simply tells us *what* the total is, whereas data labeled on the word level tells us *where* it is. The former type of data is plentiful, produced naturally and is hard to learn from. The latter is scarce, must be explicitly produced for the purpose of machine learning, and is easier to learn from.

In this paper we propose an end-to-end deep neural network architecture that can be trained directly from end-to-end information extraction data. This is our main contribution. We believe this architecture can be useful for many real-life information extraction tasks, where end-to-end data already exists, and word-level labeling is not feasible.

We evaluate our proposed architecture on a large diverse set of invoices. Invoices are somewhat special documents, in that documents from the same supplier often has a consistent layout, or template. Powerful methods exists for extracting information from templates, given that the template is known beforehand, but these methods generalize poorly across templates. Our proposed architecture addresses the harder task of learning a model that generalizes across document templates and as such can be used on unseen templates. This is important for invoices where the template often varies considerable across suppliers. We make the key assumption that the same structured information must be extracted from every document. For invoices this is a reasonable assumption, as invoices are fairly well defined documents.

Invoices are complex documents with considerable spatial structure, featuring both text and image modalities. The proposed architecture takes the spatial structure into account by using convolutional operations on the concatenated document text and image modalities. The text modality is represented in a principled manner by embedding the extracted text in a spatial grid. We assume the text of the document is given or extracted using an Optical Character Recognition (OCR) engine.

While this paper consider the case of invoices, the Attend, Copy, Parse framework is in no way limited to invoices. It could be used for any documents from which you are interested in extracting a fixed set of fields e.g. quarterly earning reports or meeting schedules from emails.

II. Related work

A. Pattern matching.

An intuitive approach to information extraction is to identify patterns in the unstructured data and use that to extract information. For instance, the total amount due in an invoice is typically to the right of a word that says "total", and is

typically a decimal number, a pattern which can be captured using a regular expression.

There's a rich literature that expand upon this general idea. For instance, Riloff et al. [1993] suggests an expressive pattern matching languages that can take the syntactic sentence structure into account, e.g. match the noun of a given verb keyword and Huffman [1995] proposes extracting multiple target values using a single joint syntactic and keyword based pattern match. See Muslea et al. [1999] for a survey.

For the more specific task of extracting information from business documents several works use a pattern matching approach. Schuster et al. [2013], Rusinol et al. [2013] and Cesarini et al. [2003] require users to annotate which words should be extracted for a given document template, then automatically generate patterns matching those words. At test time, these patterns generate candidate words, which are scored using heuristics. Dengel and Klein [2002], Esser et al. [2012] and Medvet et al. [2011] all use manually configured patterns based on keywords, parsing rules and positions.

Pattern matching generally works better the more homogeneous and structured the input is. The main disadvantages are that the patterns takes time and expertise to create and maintain, and often doesn't generalize across document templates.

B. Word classification.

Machine learning offers an elegant solution to deciding which words to extract. Given a dataset of documents and labels for each word, it becomes a regular classification task; given a word classify whether it should be extracted. If multiple values are to be extracted, e.g. total, date, etc. it becomes a multiclass classification task. The field of Natural Language Processing (NLP) uses this approach extensively, to perform a variety of tasks, e.g. Named Entity Recognition (NER) or part of speech tagging. See Collobert and Weston [2008] for an overview of tasks and methods.

Traditionally the machine learning practitioner would come up with a set of features for words and use a shallow classifier, e.g. logistic regression, SVMs, etc. Many of the insights and heuristics used in the pattern matching approach can be repurposed as features. However, state-of-the-art deep learning methods generally avoid feature engineering and favor word embeddings [Mikolov et al., 2013, Pennington et al., 2014] and deep neural networks [Ma and Hovy, 2016, Lample et al., 2016, Santos and Guimaraes, 2015].

The main drawback of the word classification approach to information extraction is the need for a richly labeled dataset as every word must be labeled. Manual labeling is an expensive process, that is consequently not feasible for many real life information extraction tasks.

Distant supervision [Mintz et al., 2009] proposes to generate the word-level labels heuristically from the available data. For instance, in our invoice example, if we know the total is `"200.00"` we can search for this string in the PDF, and label all words that match as the total. Palm et al. [2017] takes this approach and achieves state-of-the-art results on a large diverse set of invoices. The drawback is that the quality of the labels depend entirely on the quality of the manually created heuristics. The heuristics should be smart enough to know that the 200 in the string `"total: 200 $"` is probably the total, whereas 200 in the string `"200 liters"` is probably not. This is further complicated if the target string is not present letter-to-letter in the inputs.

C. End-to-end methods

Deep neural networks and the end-to-end training paradigm have lead to breakthroughs in several domains e.g. image recognition [Krizhevsky et al., 2012], and machine translation [Bahdanau et al., 2014]. We are broadly inspired by these successes to investigate end-to-end learning for information extraction from documents.

Convolutional neural networks have been used extensively on document images, e.g. segmenting documents [Yang et al., 2017, Chen et al., 2017a, Wick and Puppe, 2018], spotting handwritten words [Sudholt and Fink, 2016], classifying documents [Kang et al., 2014] and more broadly detecting text in natural scenes [Liao et al., 2017, Borisyuk et al., 2018]. In contrast to our task, these are trained on explicitly labeled datasets with information on *where* the targets are, e.g. pixel level labels, bounding boxes, etc.

Yang et al. [2017] proposes to combine the document image modality with a text embedding modality in a convolutional network for image segmentation, by fusing the modalities late in the network. We fuse the modalities as early as possible, which we find work well for our application.

The idea of reading from an an external memory using an attention mechanism similar to the one proposed in Bahdanau et al. [2014] was introduced in Graves et al. [2014] and Weston et al. [2014], Sukhbaatar et al. [2015]. Our memory implementation largely follows this paradigm, although it is read-only. External memories has since been studied extensively [Miller et al., 2016, Santoro et al., 2016, Graves et al., 2016].

III. Methods

We are given a dataset of N samples, each consisting of

- A document image $x \in [0,1]^{H \times W \times 3}$, where H and W is the height and width of the document image respectively.
- A set of P words $\mathbf{w} = \{(w_1, p_1), ..., (w_P, p_P)\}$ where w_i is the word text and $p_i \in [0,1]^4$ denotes the normalized word position and size in the document image. This would typically be the output of an OCR engine applied to the document image.
- Target strings t_k for K values we wish to extract, e.g. "2018-07-23" for a date.

The task is to learn a system that extracts the correct K output strings $\mathbf{y} = [y_1, ..., y_K]$ for a new input $(x, \mathbf{w})$. The system should be able to handle:

- Unseen document templates, i.e. generalize across document templates.
- Normalized target strings. Some target strings t_k, e.g. date and amounts, are normalized to standard formats in the end-to-end data, so may not be present letter-to-letter in the inputs.

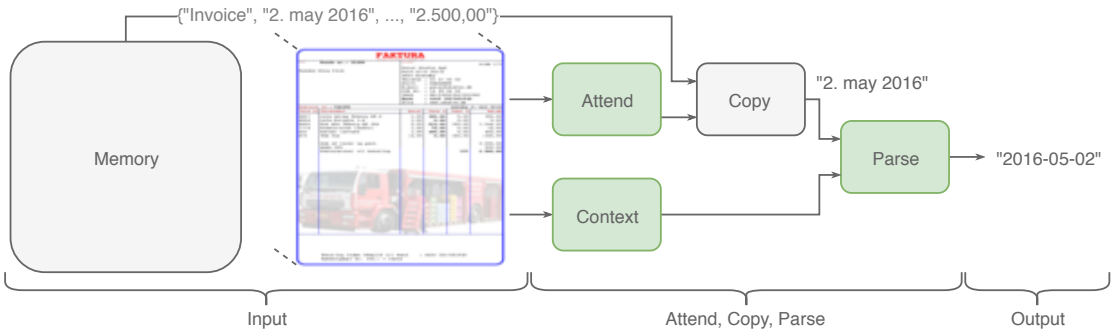


Fig. 1. Overview of the Attend, Copy, Parse architecture. All modules are end-to-end differentiable. The modules highlighted in green are learned. The document image is 128×128 pixels, the same resolution the network sees. The dashed lines indicates additional feature channels, primarily text embeddings. See equation 3 for a list of all the features.

- Target strings spanning multiple words in the input. A single normalized target string, e.g. the date, `"2018-07-23"` might be represented as several words in the input e.g. `"23rd" "July," "2018"`.
- Optional outputs. An output might be optional.

A natural human approach to information extraction is: for each value to extract: 1) locate the string to extract 2) parse it into the desired format. Our proposed framework is broadly inspired by this approach. On a high level the Attend, Copy, Parse framework produces an output string y_k as

$$\begin{aligned} a_k &= \text{Attend}_k\left(x, \mathbf{w}\right) \\ c_k &= \text{Copy}_k\left(a_k, \mathbf{w}\right) \\ h_k &= \text{Context}_k\left(x, \mathbf{w}\right) \\ y_k &= \text{Parse}_k\left(c_k, h_k\right) \ . \end{aligned}$$

The main idea is to build an external memory bank the same size as the document image, containing the words encoded as a sequence of characters at the memory positions corresponding to the positions of the words in the image. The attend module attends to this memory bank and the copy module copies out the attended string. The parse module parses the attended string into the desired output format, optionally given a context vector computed from the inputs. See figure 1 for an overview.

We will now describe each module in detail for a single model. The k subscript is dropped in the following since we train separate models for each of the K fields.

A. External memory

We start by constructing the external memory bank, $M \in \{0,1\}^{H \times W \times G \times L \times D}$ containing N-grams up to length G. In our experiments $G = 4$. The N-grams are created by an algorithm that divides the document into a number of non-intersecting lines and sorts the words by their horizontal position inside each line.

The N-grams are encoded as a sequence of one-hot encoded characters, such that L is the maximum sequence length we consider and D is the size of the character set we're using. For our experiments $L = 128$ and $D = 103$. The memory has $W \times H \times G$ slots, that can each contain an encoded N-gram. The first two dimensions correspond to the spatial dimensions of the document and the third to the length of the N-gram. This memory tensor quickly becomes too big to keep in memory. However, since it is very sparse it can be represented as a sparse tensor.

It is not immediately obvious which slots in the memory should contain the N-grams, since each N-gram span multiple pixels in the document image. We found that it suffices to store the encoded N-grams in the single slot corresponding to the top-left corner position of the first word in each N-gram. This makes the sums in the copy module considerably faster.

B. Attend

We compute unnormalized attention logits $u \in \mathbb{R}^{H \times W \times G}$ for each slot in the external memory.

$$u = \text{Attend}\left(x, \mathbf{w}\right) \ . \tag{1}$$

Since we know which slots in the memory are not empty, we only want the network to put probability mass here. To achieve this we set u to -1000 everywhere else before we compute the attention distribution $a \in [0,1]^{H \times W \times G}$ using the softmax operation.

$$a_{ijg} = \frac{e^{u_{ijg}}}{\sum_{i=1}^{H}\sum_{j=1}^{W}\sum_{g=1}^{G} e^{u_{ijg}}} \ . \tag{2}$$

In our experiments we parameterize the Attend function in the following way. We construct an input representation of the document $r \in \mathbb{R}^{H \times W \times U}$, where U is the number of feature channels.

$$r = \text{Concat}\left(x, q_w, q_p, q_c, z, \delta_x, \delta_y, \eta\right) \ , \tag{3}$$

where q_w, q_p and q_c are learned 32 dimensional word, pattern and character embeddings, respectively. The pattern embedding is an embedding of the word after all characters have been replaced with the character `x`, all digits with `0` and all

other characters with a dot. Word and pattern embeddings are replicated across the spatial extent of the entire word. Character embeddings are replicated across the spatial extent of each character in the word. In case characters overlap due to downsampling of the document image, the rightmost character wins. z is two binary indicator channels whether the N-gram at this position parses as an amount or a date, according to two pre-configured parsers. δ_x and δ_y contain the normalized ([0,1]) horizontal and vertical positions. Finally η is a binary indicator whether the external memory at this spatial position is non-empty.

We pass r through four dilated convolution blocks [Yang et al., 2017]. A dilated convolution block consists of a number of dilated convolution operations, each with a different dilation rate, that all operate in parallel on the input, and whose outputs are concatenated channel wise. Each dilated convolution block contains 4 dilated convolution operations with dilation rates $[1, 2, 4, 8]$, each with 32 3×3 filters with ReLU nonlinearities. The output of each dilated convolution block has 128 channels. See figure 2.

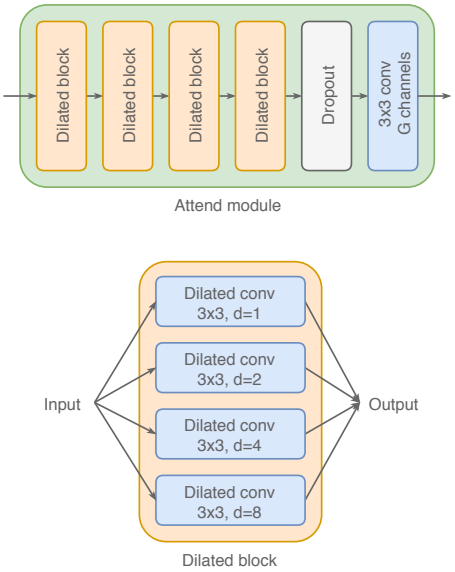


Fig. 2. Attend module and dilated block details. "d" denotes the dilation rate.

The dilated convolution block allows the network to preserve detailed local information through the layers with smaller dilation rates, while capturing a large context through the layers with higher dilation rates, both of which are important for our task.

After the 4 dilated convolution blocks we apply dropout, and a final convolution operation with G linear 3×3 filters, to get the unnormalized attention logits u, for each memory slot. We tried several other attention module architectures including U-nets [Ronneberger et al., 2015], residual networks [He et al., 2016] and deeper, wider variants of the residual blocks. The architecture described was chosen based on its performance on a validation set.

C. Copy

Given the memory M and the attention distribution we copy out the attended N-gram $c \in [0, 1]^{L \times D}$,

$$c_{ld} = \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{g=1}^{G} a_{ijg} M_{ijgld} \ . \tag{4}$$

We use the term copy, although it is a soft differentiable approximation to copy, such that each c is a weighted sum over all the N-grams present in the document. Accordingly, each character c_l is a distribution over all the characters in the character set.

D. Context

The context of an N-gram is often important for parsing it correctly. For instance the date "02/03/2018" is ambiguous. It can either be the 2nd of March or the 3rd of February, but the context can disambiguate it. For instance, if the language of the document is German, then it is more likely that it is the former. In our experiments we use the following context function.

$$h_f = \sum_{i=1}^{H} \sum_{j=1}^{W} \sum_{g=1}^{G} a_{ijg} v_{ijf} \ ,$$

where $v \in \mathbb{R}^{H \times W \times 128}$ is the output of the last dilated convolution block of the attend module, after dropout. Thus, h is a vector $h \in \mathbb{R}^{128}$. For simplicity the implementation of the context function in our experiments only depend on the attention module, but in general, it can be any function of the input $(x, \mathbf{w})$.

E. Parse

Given the attended word c, and the context vector h, we parse c into the output y. This is in essence a character based sequence to sequence task:

$$y = \text{Parse}(c, h) \ . \tag{5}$$

The implementation of Parse depends on the output format. Some fields might not need any parsing, while dates and amounts need different parsing. In the following we describe the four different parsers we use.

NoOp Parser. This is the simplest parser. It returns the attended sequence as is

$$y = c \ .$$

Optional Parser. This returns a mixture of the attended input and a string ϵ consisting solely of `<EOS>` tokens

$$\alpha = f(h)$$
$$y = (1 - \alpha)c + \alpha\epsilon \ ,$$

where α in $[0, 1]$ is the mixture parameter. We use a single fully connected layer with one sigmoid output unit for f.

Date Parser. Our target dates are represented in ISO 8601 format, e.g. "2018-07-23" which has a constant output length of ten characters, with two fixed characters. We use the following architecture in our experiments which we find work well:

$$e = \sum_{l=1}^{L^*} \text{CNNMP}(c)$$
$$y = \text{MLP}\left(\text{Concat}(e, h)\right) \ ,$$

where CNNMP is 4 layers of CNN with 128 ReLU units with kernel size 3, followed by stride 2 maxpooling, $L^* = \frac{L}{2^4}$ is the length of the sequence after the four maxpooling layers. As such $e \in \mathbb{R}^{128}$. MLP is 3 fully connected layers with 128 ReLU units, followed by dropout, and finally a linear layer with $10 \times D$ linear outputs, which outputs the unnormalized logits for the ten characters. We could use a smaller output character set than D, but for simplicity we use a single character set for representing all characters.

Amount Parser. The amounts in the dataset are normalized by removing leading and trailing zeros, e.g. "00.020" gets formatted to "0.02" and "1.00" to "1". We use a fixed output size of 16 characters.

Our amount parser is a pointer-generator networks [See et al., 2017], with a bidirectional LSTM with 128 units to encode the input sequence, and a LSTM with 128 units to generate the output hidden states. The idea behind using a pointer-generator network is that if the attended input is a digit, it can be copied directly, if it is a decimal separator, the network can generate a dot, and if it is the last significant digit the network can generate an <EOS> token. For details see the appendix.

F. Loss and attention regularization

The main loss is the average cross-entropy between the targets characters t and the output characters y.

We found that the softmax operation in equation (2) had a tendency to underflow for many of the non-empty memory positions causing the network to get stuck in the initial phase of training. To solve this we added a regularization term to the loss defined as the cross-entropy between a uniform attention distribution over the non-empty memory positions and the attention distribution produced by the attend module in equation (2):

$$\mathcal{L}(y, t) = -\frac{1}{L}\sum_{l=1}^{L} \log(y_l[t_l]) - \lambda \frac{1}{|I|} \sum_{(i,j,g)\in I} \log(a_{ijg}) \ ,$$

where $y_l[t_l]$ indicates the t_lth element of y_l, I is the set of non-empty memory positions and λ is a small scalar hyper-parameter. The regularization encourages the network to place attention mass uniformly on the non-empty memory positions.

IV. Experiments

Our dataset consists of 1,181,990 invoices from 43,023 different suppliers. The dataset is obtained from production usage of an invoice information extraction system. The suppliers upload a document image and the system extracts several fields of interest. If the system makes a mistake for a field the suppliers themselves type in the correct value for the field. Note, the suppliers do not need to indicate which word in the document corresponds to the field so as to not burden them with additional labeling effort. As such the dataset fits our end-to-end dataset definition and the description in the methods section; it is not known which words in the document corresponds to the fields we wish to extract, we simply know the target value of each field. We focus on seven important fields, 1) the invoice number, 2) the order number, 3) the date, 4) the total, 5) the sub total before tax, 6) the tax total and 7) the tax percent.

We split the suppliers randomly into 42,163 training suppliers and 860 testing suppliers. The training and testing sets of documents consists of all documents from each set of suppliers, 1,153,078 and 28,912 documents, respectively. Splitting on the suppliers instead of splitting on the documents allows us to measure how well the model generalize across document templates, assuming all suppliers use different templates.

There are two main sources of noise in this dataset, that will limit the maximum performance of any system.

1) OCR. If the OCR engine makes a mistake, it is very hard to recover from that later on. With a powerful enough parser it is not impossible, but still unlikely.
2) Suppliers. The suppliers can and will type in field values that are not present in the document image. For instance the supplier might type in the date they are correcting the document, rather than the date present in the document, in order to not backdate the invoice. This kind of noise is near impossible to correct for.

In order to estimate the maximum performance of a system for this dataset, we perform the following analysis. Given a target value t_k we try to find this value in the document. If t_k is an amount, we use a recall oriented amount parser based on regular expressions to parse the strings in the document. We apply a similar procedure for dates. This way we can measure an approximate fraction of target strings present in the document. We use the term "Readable" for this fraction of target values. For the fields that do not use a parser, i.e, the invoice number and the order number, this is an upper bound on the possible accuracy for our proposed end-to-end architecture. For the other fields, it is conceivable that we can learn a better parser than the regular expression based parsers, but it is still a decent approximation of the achievable accuracy.

We compare the proposed end-to-end system to the production system described in Palm et al. [2017], which is trained and tested on the same data. In short the production system uses the word-classification approach with word labels derived using a distant supervision heuristic. A logistic regression classifier classifies each word into, e.g. the invoice number, total, etc. After classifying the words, they are parsed, and the system uses heuristics to pick the best candidate words, e.g. for the total fields, it tries to find a joint solution such that the totals add up, etc.

We train a separate model for each of the seven fields, which all have the same architecture and hyper-parameters, except for the parsers. The invoice number uses the no-op parser, the order number the optional parser, the date the date parser, and the rest use the amount parser. We pre-train the amount parser on amounts extracted from the training documents, parsed with a conventional regular expression based parser. We observe that the amount parser quickly learns to replicate the regular expressions. Each model is optimized using the Adam [Kingma and Ba, 2014] optimizer, with a batch size of 32, learning rate 0.0003, dropout of 0.5, $\lambda = 0.0001$, and trained for 50,000 batch updates, with a small L2 regularization of 0.0001. We resize the document image to 128×128 pixels, disregarding aspect ratio. See table I for the results.

TABLE I
RESULTS. FRACTION OF CORRECT VALUES.

Field	Readable	Prod	Prod-	Attend, Copy, Parse
Number	0.90	0.78	0.78	**0.87**
Order id	0.90	0.82	0.82	**0.84**
Date	0.83	0.70	0.70	**0.80**
Total	0.81	**0.85**	0.77	0.81
Sub total	0.84	**0.84**	0.73	0.79
Tax total	0.80	**0.87**	0.77	0.80
Tax percent	0.79	0.83	0.68	**0.87**
Average	0.84	0.81	0.75	**0.83**

The proposed Attend, Copy, Parse system performs better on average, and close to the approximate maximum accuracy possible given the architecture, denoted "Readable". The invoice number is a good field to compare how good the respective systems are purely at "finding" the correct N-gram, since there's no parsing involved in either system. Here the Attend, Copy, Parse system excels. However, the production system, "Prod", performs significantly better on 3 of the 4 amount fields. The production system uses a heuristic to pick the total fields jointly, such that they add up. In order to test how much this heuristic improves the results we test a version of the production system with this heuristic disabled. We denote this "Prod-". This version simply choose the word with the highest probability given by the logistic regression classifier that can be parsed into an amount for each of the total fields. It is clear from the results that this heuristic considerably boosts the accuracy on the total fields. Interestingly the Attend, Copy, Parse architecture recovers more correct tax percentages than can be found in the documents. Upon closer inspection this is because many documents have zero taxes, but do not contain an explicit zero. The pointer-generator amount parser learns to generate a zero in these cases.

V. DISCUSSION

We have presented a deep neural network architecture for end-to-end information extraction from documents. The architecture does not need expensive word-level labels, instead it can directly use the end-to-end data that is naturally produced in a human information extraction tasks. We evaluated the proposed architecture on a large diverse set of invoices, where we outperform a state-of-the-art production system based on distant supervision, word classification and heuristics.

The proposed architecture can be improved in several ways. The most obvious shortcoming is that it can only handle single page documents. This is theoretically easy to remedy by adding a new page dimension to the inputs, turning the spatial convolutions into volumetric convolutions, and letting the attend module output attention over the pages as well. The main concern is the computational resources required for this change.

It should be possible to learn a single network which output the K strings. We experimented with this, by letting the attention module output K attention distributions, having K separate copy and parse modules, and training everything jointly using the sum of losses across each of the K outputs. It worked, but less well. We suspect it is because of imbalance between the losses. For instance dates have lower entropy in general compared to invoice numbers. Loss imbalance is a general challenge in the multi-task learning setting [Kendall et al., 2017, Chen et al., 2017b].

Going from not taking the dependencies between the total fields into account (Prod-) to taking them into account (Prod) significantly increases the performance of the system even given a relatively weak classifier. Unfortunately, the heuristic used in the Prod system cannot be directly used with the Attend, Copy, Parse architecture as it is not differentiable.

We did come up with an idea to incorporate the two constraints (total = sub total + tax total and tax total = sub total $\cdot$ tax percentage) in our Attend, Copy, Parse framework but it did not improve on the results. Here we describe the idea briefly. It consists of three steps: 1) let the network output a probability that each total field should be inferred from the constraints instead of being outputted directly, 2) Assuming you will sample which fields to infer from this distribution, write up the marginal probability of all the fields being correct and 3) Use the negative log of this probability as a loss instead of the four individual total field losses. If you sample three or four fields to infer, then the probability of all the fields being correct is zero, since you can at most infer two of the fields from the constraints. If you sample two fields or less, then the probability that all the fields are correct is the probability that all the non-inferred fields are correct. The marginal probability that all of the fields are correct is then the sum over the probability that a permutation of fields to be inferred is chosen, multiplied by the probability that all the non-inferred fields for the given permutation are correct. There's only $\binom{4}{0} + \binom{4}{1} + \binom{4}{2} = 11$ permutations that give non-zero probabilities, so they can simply be computed and summed.

The presented architecture can only extract non-recurring fields as opposed to recurring, structured fields such as invoice lines. Theoretically it should be possible to output the lines by recurrently outputting the fields of a single line at a time, and then conditioning the attention module on the previously outputted line. This would be an interesting direction for future work.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Fedor Borisyuk, Albert Gordo, and Viswanath Sivakumar. Rosetta: Large scale system for text detection and recognition in images. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 71–79. ACM, 2018.

Francesca Cesarini, Enrico Francesconi, Marco Gori, and Giovanni Soda. Analysis and understanding of multi-class invoices. *Document Analysis and Recognition*, 6(2):102–114, 2003.

Kai Chen, Mathias Seuret, Jean Hennebert, and Rolf Ingold. Convolutional neural networks for page segmentation of historical document images. In *Document Analysis and Recognition (ICDAR), 2017 14th IAPR International Conference on*, volume 1, pages 965–970. IEEE, 2017a.

Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv preprint arXiv:1711.02257*, 2017b.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

Andreas R Dengel and Bertin Klein. smartfix: A requirements-driven system for document analysis and understanding. In *International Workshop on Document Analysis Systems*, pages 433–444. Springer, 2002.

Daniel Esser, Daniel Schuster, Klemens Muthmann, Michael Berger, and Alexander Schill. Automatic indexing of scanned documents: a layout-based approach. In *Document Recognition and Retrieval XIX*, volume 8297, page 82970H. International Society for Optics and Photonics, 2012.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Scott B Huffman. Learning information extraction patterns from examples. In *International Joint Conference on Artificial Intelligence*, pages 246–260. Springer, 1995.

Le Kang, Jayant Kumar, Peng Ye, Yi Li, and David Doermann. Convolutional neural networks for document image classification. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 3168–3172. IEEE, 2014.

Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *arXiv preprint arXiv:1705.07115*, 3, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

Minghui Liao, Baoguang Shi, Xiang Bai, Xinggang Wang, and Wenyu Liu. Textboxes: A fast text detector with a single deep neural network. In *AAAI*, pages 4161–4167, 2017.

Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*, 2016.

Eric Medvet, Alberto Bartoli, and Giorgio Davanzo. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(4):335–347, 2011.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *arXiv preprint arXiv:1606.03126*, 2016.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

Ion Muslea et al. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 workshop on machine learning for information extraction*, volume 2. Orlando Florida, 1999.

Rasmus Berg Palm, Ole Winther, and Florian Laws. Cloudscan-a configuration-free invoice analysis system using recurrent neural networks. *arXiv preprint arXiv:1708.07403*, 2017.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Ellen Riloff et al. Automatically constructing a dictionary for information extraction tasks. In *AAAI*, volume 1, pages 2–1. Citeseer, 1993.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-

net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

Marçal Rusinol, Tayeb Benkhelfallah, and Vincent Poulain dAndecy. Field extraction from administrative documents by incremental structural templates. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1100–1104. IEEE, 2013.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

Cicero Nogueira dos Santos and Victor Guimaraes. Boosting named entity recognition with neural character embeddings. *arXiv preprint arXiv:1505.05008*, 2015.

Daniel Schuster, Klemens Muthmann, Daniel Esser, Alexander Schill, Michael Berger, Christoph Weidling, Kamil Aliyev, and Andreas Hofmeier. Intellix–end-user trained information extraction for document archiving. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 101–105. IEEE, 2013.

Abigail See, Peter J Liu, and Christopher D Manning. Get to the point: Summarization with pointer-generator networks. *arXiv preprint arXiv:1704.04368*, 2017.

Sebastian Sudholt and Gernot A Fink. Phocnet: A deep convolutional neural network for word spotting in handwritten documents. In *Frontiers in Handwriting Recognition (ICFHR), 2016 15th International Conference on*, pages 277–282. IEEE, 2016.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, 2014. URL `http://arxiv.org/abs/1410.3916`.

Christoph Wick and Frank Puppe. Fully convolutional neural networks for page segmentation of historical document images. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 287–292. IEEE, 2018.

Xiao Yang, Ersin Yumer, Paul Asente, Mike Kraley, Daniel Kifer, and C Lee Giles. Learning to extract semantic structure from documents using multimodal fully convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

VI. Appendix

A. Amount parser details

We use a fixed output size of $O = 16$ characters. Each output character $y_o \in [0,1]^D$ is a weighted sum over characters from the attended string c, or generated from the character set.

$$y_o = \rho_o \sum_{l=1}^{L} (a_{ol} c_l) + (1 - \rho_o) g(e_o) \ ,$$

where $\rho_o \in [0,1]$ determines whether y_o should be copied from the attended string c or generated from the character set. $a \in [0,1]^{O \times L}$ is O attention distributions over each character in the input string; one for each character in the output string, which determines which character to copy. g is a function that maps $e_o \in \mathbb{R}^{256}$ to a distribution over the character set. In our experiments we use a single dense layer with D outputs and a softmax nonlinearity. ρ_o is given as

$$\rho_o = f(e_o) \ ,$$

where f is a function that maps e_o to $[0,1]$. In our experiments we used a single dense layer with a single sigmoid output. e_o is

$$e_o = \sum_{l=1}^{L} (a_{ol} h_l) \ ,$$

where $h \in \mathbb{R}^{L \times 256}$ is the encoded input string c. We use a bidirectional LSTM with 128 hidden units each to encode the input string,

$$h = \text{BiLSTM}(c_k) \ .$$

All that remains to be defined are a, the O attention distributions over the input characters.

$$a_{ol} = \frac{e^{\alpha_{ol}}}{\sum_{l=1}^{L} e^{\alpha_{ol}}}$$
$$\alpha_{ol} = \phi(u_o, h_l) \ ,$$

where ϕ maps $u_o \in \mathbb{R}^{128}$ and $h_l \in \mathbb{R}^{256}$ to R. Following Bahdanau et al. [2014] we use

$$\phi(u_o, h_l) = \tanh(u_o W_u + h_l W_h) W_t \ ,$$

where tanh is applied element-wise and $W_u \in \mathbb{R}^{128 \times 128}$, $W_h \in \mathbb{R}^{256 \times 128}$ and $W_t \in \mathbb{R}^{128 \times 1}$ are learned weight matrices. Finally, $u \in \mathbb{R}^{O \times 128}$ is the output of a 128 unit LSTM run for O steps with no inputs.

APPENDIX D

Recurrent Relational Networks

Published at Conference on Neural Information Processing Systems (NIPS) 2018

Recurrent Relational Networks

Rasmus Berg Palm
Technical University of Denmark
Tradeshift
rapal@dtu.dk

Ulrich Paquet
DeepMind
upaq@google.com

Ole Winther
Technical University of Denmark
olwi@dtu.dk

Abstract

This paper is concerned with learning to solve tasks that require a chain of interdependent steps of relational inference, like answering complex questions about the relationships between objects, or solving puzzles where the smaller elements of a solution mutually constrain each other. We introduce the *recurrent relational network*, a general purpose module that operates on a *graph* representation of objects. As a generalization of Santoro et al. [2017]'s relational network, it can augment any neural network model with the capacity to do many-step relational reasoning. We achieve state of the art results on the bAbI textual question-answering dataset with the recurrent relational network, consistently solving 20/20 tasks. As bAbI is not particularly challenging from a relational reasoning point of view, we introduce Pretty-CLEVR, a new diagnostic dataset for relational reasoning. In the Pretty-CLEVR set-up, we can vary the question to control for the number of relational reasoning steps that are required to obtain the answer. Using Pretty-CLEVR, we probe the limitations of multi-layer perceptrons, relational and recurrent relational networks. Finally, we show how recurrent relational networks can learn to solve Sudoku puzzles from supervised training data, a challenging task requiring upwards of 64 steps of relational reasoning. We achieve state-of-the-art results amongst comparable methods by solving 96.6% of the hardest Sudoku puzzles.

1 Introduction

A central component of human intelligence is the ability to abstractly reason about objects and their interactions [Spelke et al., 1995, Spelke and Kinzler, 2007]. As an illustrative example, consider solving a Sudoku. A Sudoku consists of 81 cells that are arranged in a 9-by-9 grid, which must be filled with digits 1 to 9 so that each digit appears exactly once in each row, column and 3-by-3 non-overlapping box, with a number of digits given [1]. To solve a Sudoku, one methodically reasons about the puzzle in terms of its cells and their interactions over many steps. One tries placing digits in cells and see how that affects other cells, iteratively working toward a solution.

Contrast this with the canonical deep learning approach to solving problems, the multilayer perceptron (MLP), or multilayer convolutional neural net (CNN). These architectures take the entire Sudoku as an input and output the entire solution in a single forward pass, ignoring the inductive bias that objects exists in the world, and that they affect each other in a consistent manner. Not surprisingly these models fall short when faced with problems that require even basic relational reasoning [Lake et al., 2016, Santoro et al., 2017].

The relational network of Santoro et al. [2017] is an important first step towards a simple module for reasoning about objects and their interactions but it is limited to performing a single relational operation, and was evaluated on datasets that require a maximum of three steps of reasoning (which,

[1] We invite the reader to solve the Sudoku in the supplementary material to appreciate the difficulty of solving a Sudoku in which 17 cells are initially filled.

Preprint. Work in progress.

surprisingly, can be solved by a single relational reasoning step as we show). Looking beyond relational networks, there is a rich literature on logic and reasoning in artificial intelligence and machine learning, which we discuss in section 5.

Toward generally realizing the ability to methodically reason about objects and their interactions over many steps, this paper introduces a composite function, the *recurrent relational network*. It serves as a modular component for many-step relational reasoning in end-to-end differentiable learning systems. It encodes the inductive biases that 1) objects exists in the world 2) they can be sufficiently described by properties 3) properties can change over time 4) objects can affect each other and 5) given the properties, the effects object have on each other is *invariant to time*.

An important insight from the work of Santoro et al. [2017] is to decompose a function for relational reasoning into two components or "modules": a perceptual front-end, which is tasked to recognize objects in the raw input and represent them as vectors, and a relational reasoning module, which uses the representation to reason about the objects and their interactions. Both modules are trained jointly end-to-end. In computer science parlance, the relational reasoning module implements an *interface*: it operates on a graph of nodes and directed edges, where the nodes are represented by real valued vectors, and is differentiable. This paper chiefly develops the relational reasoning side of that interface.

Some of the tasks we evaluate on can be efficiently and perfectly solved by hand-crafted algorithms that operate on the symbolic level. For example, 9-by-9 Sudokus can be solved in a fraction of a second with constraint propagation and search [Norvig, 2006] or with dancing links [Knuth, 2000]. These symbolic algorithms are superior in every respect but one: they don't comply with the interface, as they are not differentiable and don't work with real-valued vector descriptions. They therefore cannot be used in a combined model with a deep learning perceptual front-end and learned end-to-end.

Following Santoro et al. [2017], we use the term "relational reasoning" liberally for an object- and interaction-centric approach to problem solving. Although the term "relational reasoning" is similar to terms in other branches of science, like relational logic or first order logic, no direct parallel is intended.

This paper considers many-step relational reasoning, a challenging task for deep learning architectures. We develop a recurrent relational reasoning module, which constitutes our main contribution. We show that it is a powerful architecture for many-step relational reasoning on three varied datasets, achieving state-of-the-art results on bAbI and Sudoku.

2 Recurrent Relational Networks

We ground the discussion of a recurrent relational network in something familiar, solving a Sudoku puzzle. A simple strategy works by noting that if a certain Sudoku cell is given as a "7", one can safely remove "7" as an option from other cells in the same row, column and box. In a message passing framework, that cell needs to send a message to each other cell in the same row, column, and box, broadcasting it's value as "7", and informing those cells not to take the value "7". In an iteration t, these messages are sent simultaneously, in parallel, between all cells. Each cell i should then consider all incoming messages, and update its internal state h_i^t to h_i^{t+1}. With the updated state each cell should send out new messages, and the process repeats.

Message passing on a graph. The recurrent relational network will learn to pass messages on a *graph*. For Sudoku, the graph has $i \in \{1, 2, ..., 81\}$ nodes, one for each cell in the Sudoku. Each node has an input feature vector x_i, and edges to and from all nodes that are in the same row, column and box in the Sudoku. The *graph* is the input to the relational reasoning module, and vectors x_i would generally be the output of a perceptual front-end, for instance a convolutional neural network. Keeping with our Sudoku example, each x_i encodes the initial cell content (empty or given) and the row and column position of the cell.

At each step t each node has a hidden state vector h_i^t, which is initialized to the features, such that $h_i^0 = x_i$. At each step t, each node sends a message to each of its neighboring nodes. We define the message m_{ij}^t from node i to node j at step t by

$$m_{ij}^t = f\left(h_i^{t-1}, h_j^{t-1}\right) \ , \tag{1}$$

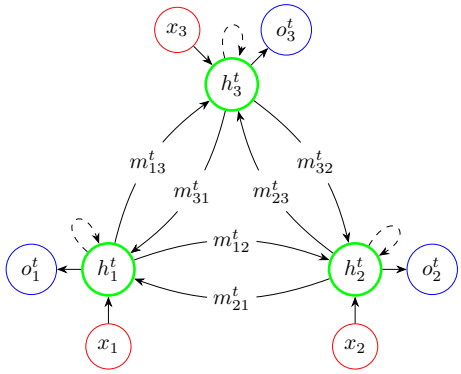


Figure 1: A *recurrent relational network* on a fully connected graph with 3 nodes. The nodes' hidden states h_i^t are highlighted with green, the inputs x_i with red, and the outputs o_i^t with blue. The dashed lines indicate the recurrent connections. Subscripts denote node indices and superscripts denote steps t. For a figure of the same graph unrolled over 2 steps see the supplementary material.

where f, the message function, is a multi-layer perceptron. This allows the network to learn what kind of messages to send. In our experiments, MLPs with linear outputs were used. Since a node needs to consider all the incoming messages we sum them with

$$m_j^t = \sum_{i \in N(j)} m_{ij}^t \,, \tag{2}$$

where $N(j)$ are all the nodes that have an edge into node j. For Sudoku, $N(j)$ contains the nodes in the same row, column and box as j. In our experiments, since the messages in (1) are linear, this is similar to how log-probabilities are summed in belief propagation [Murphy et al., 1999].

Recurrent node updates. Finally we update the node hidden state via

$$h_j^t = g\left(h_j^{t-1}, x_j, m_j^t\right) \,, \tag{3}$$

where g, the node function, is another learned neural network. The dependence on the previous node hidden state h_j^{t-1} allows the network to iteratively work towards a solution instead of starting with a blank slate at every step. Injecting the feature vector x_j at each step like this allows the node function to focus on the messages from the other nodes instead of trying to remember the input.

Supervised training. The above equations for sending messages and updating node states define a recurrent relational network's core. To train a recurrent relational network in a supervised manner to solve a Sudoku we introduce an output probability distribution over the digits 1-9 for each of the nodes in the graph. The output distribution o_i^t for node i at step t is given by

$$o_i^t = r\left(h_i^t\right) \,, \tag{4}$$

where r is a MLP that maps the node hidden state to the output probabilities, e.g. using a softmax nonlinearity. Given the target digits $\mathbf{y} = \{y_1, y_2, ..., y_{81}\}$ the loss at step t, is then the sum of cross-entropy terms, one for each node: $l^t = -\sum_{i=1}^{I} \log o_i^t[y_i]$, where $o_i[y_i]$ is the y_i'th component of o_i. Equations (1) to (4) are illustrated in figure 1.

Convergent message passing. A distinctive feature of our proposed model is that we minimize the cross entropy between the output and target distributions at *every step*.

At test time we only consider the output probabilities at the last step, but having a loss at every step during training is beneficial. Since the target digits y_i are constant over the steps, it encourages the network to learn a convergent message passing algorithm. Secondly, it helps with the vanishing gradient problem.

Variations. If the edges are unknown, the graph can be assumed to be fully connected. In this case the network will need to learn which objects interact with each other. If the edges have attributes, e_{ij}, the message function in equation 1 can be modified such that $m_{ij}^t = f\left(h_i^{t-1}, h_j^{t-1}, e_{ij}\right)$. If the output of interest is for the whole graph instead of for each node the output in equation 4 can be modified such that there's a single output $o^t = r\left(\sum_i h_i^t\right)$. The loss can be modified accordingly.

3 Experiments

Code to reproduce all experiments can be found at github.com/rasmusbergpalm/recurrent-relational-networks.

3.1 bAbI question-answering tasks

Table 1: bAbI results. Trained jointly on all 20 tasks using the 10,000 training samples. Entries marked with an asterix are our own experiments, the rest are from the respective papers.

Method	N	Mean Error (%)	Failed tasks (err. >5%)
*RRN** (this work)	15	$\mathbf{0.46 \pm 0.77}$	$\mathbf{0.13 \pm 0.35}$
SDNC [Rae et al., 2016]	15	6.4 ± 2.5	4.1 ± 1.6
DAM [Rae et al., 2016]	15	8.7 ± 6.4	5.4 ± 3.4
SAM [Rae et al., 2016]	15	11.5 ± 5.9	7.1 ± 3.4
DNC [Rae et al., 2016]	15	12.8 ± 4.7	8.2 ± 2.5
NTM [Rae et al., 2016]	15	26.6 ± 3.7	15.5 ± 1.7
LSTM [Rae et al., 2016]	15	28.7 ± 0.5	17.1 ± 0.8
EntNet [Henaff et al., 2016]	5	9.7 ± 2.6	5 ± 1.2
ReMo [Yang et al., 2018]	1	1.2	1
RN [Santoro et al., 2017]	1	N/A	2
MemN2N [Sukhbaatar et al., 2015]	1	7.5	6

bAbI is a text based QA dataset from Facebook [Weston et al., 2015] designed as a set of prerequisite tasks for reasoning. It consists of 20 types of tasks, with 10,000 questions each, including deduction, induction, spatial and temporal reasoning. Each question, e.g. "Where is the milk?" is preceded by a number of facts in the form of short sentences, e.g. "Daniel journeyed to the garden. Daniel put down the milk." The target is a single word, in this case "garden", one-hot encoded over the full bAbI vocabulary of 177 words. A task is considered solved if a model achieves greater than 95% accuracy. The most difficult tasks require reasoning about three facts.

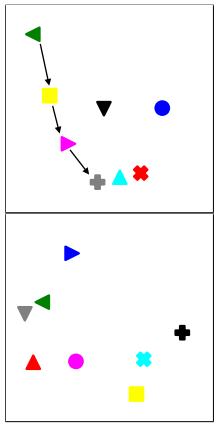
To map the questions into a graph we treat the facts related to a question as the nodes in a fully connected graph up to a maximum of the last 20 facts. The fact and question sentences are both encoded by Long Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997] layers with 32 hidden units each. We concatenate the last hidden state of each LSTM and pass that through a MLP. The output is considered the node features x_i. Following [Santoro et al., 2017] all edge features e_{ij} are set to the question encoding. We train the network for three steps. At each step, we sum the node hidden states and pass that through a MLP to get a single output for the whole graph. For details see the supplementary material.

Our trained network solves 20 of 20 tasks in 13 out of 15 runs. This is state-of-the-art and markedly more stable than competing methods. See table 1. We perform ablation experiment to see which parts of the model are important, including varying the number of steps. We find that using dropout and appending the question encoding to the fact encodings is important for the performance. See the supplementary material for details.

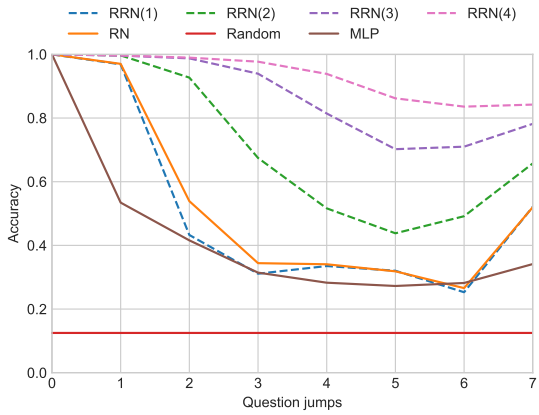
Surprisingly, we find that we only need a single step of relational reasoning to solve all the bAbI tasks. This is surprising since the hardest tasks requires reasoning about three facts. It's possible that there are superficial correlations in the tasks that the model learns to exploit. Alternatively the model learns to compress all the relevant fact-relations into the 128 floats resulting from the sum over the node hidden states, and perform the remaining reasoning steps in the output MLP. Regardless, it appears multiple steps of relational reasoning are not important for the bAbI dataset.

3.2 Pretty-CLEVR

Given that bAbI did not require multiple steps of relational reasoning and in order to test our hypothesis that our proposed model is better suited for tasks requiring more steps of relational reasoning we create a diagnostic dataset "Pretty-CLEVER". It can be seen as an extension of the "Sort-of-CLEVR" data set by [Santoro et al., 2017] which has questions of a non-relational and relational nature. "Pretty-CLEVR" takes this a step further and has non-relational questions as well as questions requiring *varying* degrees of relational reasoning.



(a) Samples.



(b) Results.

Figure 2: 2a Two samples of the Pretty-CLEVR diagnostic dataset. Each sample has 128 questions associated, exhibiting varying levels of relational reasoning difficulty. For the topmost sample the solution to the question: "green, 3 jumps", which is "plus", is shown with arrows. 2b Random corresponds to picking one of the eight possible outputs at random (colors or shapes, depending on the input). The RRN is trained for four steps but since it predicts at each step we can evaluate the performance for each step. The the number of steps is stated in parentheses.

Pretty-CLEVR consists of scenes with eight colored shapes and associated questions. Questions are of the form: "Starting at object X which object is N jumps away?". Objects are uniquely defined by their color or shape. If the start object is defined by color, the answer is a shape, and vice versa. Jumps are defined as moving to the closest object, without going to an object already visited. See figure 2a. Questions with zero jumps are non-relational and correspond to: "What color is shape X?" or "What shape is color X?". We create 100,000 random scenes, and 128 questions for each (8 start objects, 0-7 jumps, output is color or shape), resulting in 12.8M questions. We also render the scenes as images. The "jump to nearest" type question is chosen in an effort to eliminate simple correlations between the scene state and the answer. It is highly non-linear in the sense that slight differences in the distance between objects can cause the answer to change drastically. It is also asymmetrical, i.e. if the question "x, n jumps" equals "y", there is no guarantee that "y, n jumps" equals "x". We find it is a surprisingly difficult task to solve, even with a powerful model such as the RRN. We hope others will use it to evaluate their relational models.[2]

Since we are solely interested in examining the effect of multiple steps of relational reasoning we train on the state descriptions of the scene. We consider each scene as a fully connected undirected graph with 8 nodes. The feature vector for each object consists of the position, shape and color. We encode the question as the start object shape or color and the number of jumps. As we did for bAbI we concatenate the question and object features and pass it through a MLP to get the node features x_i. To make the task easier we set the edge features to the euclidean distance between the objects. We train our network for four steps and compare to a single step relational network and a baseline

[2]Pretty-CLEVR is available online as part of the code for reproducing experiments.

MLP that considers the entire scene state, all pairwise distances, and the question as a single vector. For details see the supplementary material.

Mirroring the results from the "Sort-of-CLEVR" dataset the MLP perfectly solves the non-relational questions, but struggle with even single jump questions and seem to lower bound the performance of the relational networks. The relational network solves the non-relational questions as well as the ones requiring a single jump, but the accuracy sharply drops off with more jumps. This matches the performance of the recurrent relational network which generally performs well as long as the number of steps is greater than or equal to the number of jumps. See fig 2b. It seems that, despite our best efforts, there are spurious correlations in the data such that questions with six to seven jumps are easier to solve than those with four to five jumps.

3.3 Sudoku

We create training, validation and testing sets totaling 216,000 Sudoku puzzles with a uniform distribution of givens between 17 and 34. We consider each of the 81 cells in the 9x9 Sudoku grid a node in a graph, with edges to and from each other cell in the same row, column and box. The node features x_i are the output of a MLP which takes as input the digit for the cell (0-9, 0 if not given), and the row and column position (1-9). Edge features are not used. We run the network for 32 steps and at every step the output function r maps each node hidden state to nine output logits corresponding to the nine possible digits. For details see the supplementary material.

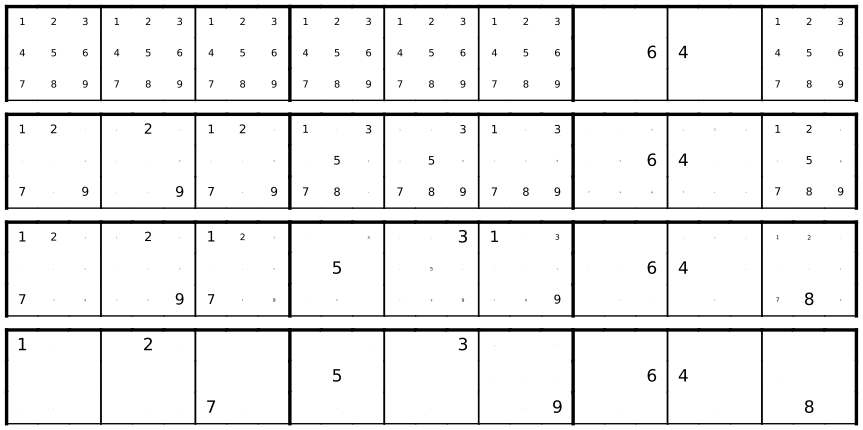


Figure 3: Example of how the trained network solves part of a Sudoku. Only the top row of a full 9x9 Sudoku is shown for clarity. From top to bottom steps 0, 1, 8 and 24 are shown. See the supplementary material for a full Sudoku. Each cell displays the digits 1-9 with the font size scaled (non-linearly for legibility) to the probability the network assigns to each digit. Notice how the network eliminates the given digits 6 and 4 from the other cells in the first step. Animations showing how the trained network solves Sodukos, including a failure case can be found at imgur.com/a/ALsfB.

Our network learns to solve 94.1% of even the hardest 17-givens Sudokus after 32 steps. We only consider a puzzled solved if all the digits are correct, i.e. no partial credit is given for getting individual digits correct. For more givens the accuracy (fraction of test puzzles solved) quickly approaches 100%. Since the network outputs a probability distribution for each step, we can visualize how the network arrives at the solution step by step. For an example of this see figure 3.

To examine our hypothesis that multiple steps are required we plot the accuracy as a function of the number of steps. See figure 4. We can see that even simple Sudokus with 33 givens require upwards of 10 steps of relational reasoning, whereas the harder 17 givens continue to improve even after 32 steps. Figure 4 also shows that the model has learned a convergent algorithm. The model was trained for 32 steps, but seeing that the accuracy increased with more steps, we ran the model for 64 steps during testing. At 64 steps the accuracy for the 17 givens puzzles increases to 96.6%.

We also examined the importance of the row and column features by multiplying the row and column embeddings by zero and re-tested our trained network. At 64 steps with 17 givens, the accuracy changed to 96.7%. It thus seems the network does not use the row and column position information to solve the task.

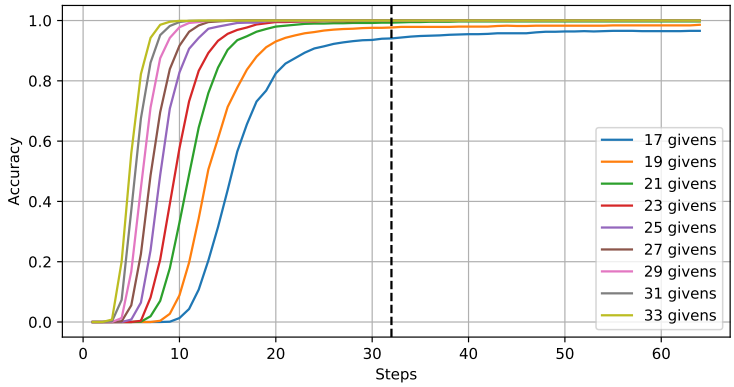


Figure 4: Fraction of test puzzles solved as a function of number of steps. Even simple Sudokus with 33 givens require about 10 steps of relational reasoning to be solved. The dashed vertical line indicates the 32 steps the network was trained for. The network appears to have learned a convergent relational reasoning algorithm such that more steps beyond 32 improve on the hardest Sudokus.

We compare our network to several other differentiable methods. See table 2. We train two relational networks: a node and a graph centric. For details see the supplementary material. Of the two, the node centric was considerably better. The node centric correspond exactly to our proposed network with a single step, yet fails to solve any Sudoku. This shows that multiple steps are crucial for complex relational reasoning. Our network outperforms loopy belief propagation, with parallel and random messages passing updates [Bauke, 2008]. It also outperforms a version of loopy belief propagation modified specifically for solving Sudokus that uses 250 steps, Sinkhorn balancing every two steps and iteratively picks the most probable digit [Khan et al., 2014]. We also compare to learning the messages in parallel loopy BP as presented in Lin et al. [2015]. We tried a few variants including a single step as presented and 32 steps with and without a loss on every step, but could not get it to solve any 17 given Sudokus. Finally we outperform Park [2016] which treats the Sudoku as a 9x9 image, uses 10 convolutional layers, iteratively picks the most probable digit, and evaluate on easier Sudokus with 24-36 givens. We also tried to train a version of our network that only had a loss at the last step. It was harder to train, performed worse and didn't learn a convergent algorithm.

Table 2: Comparison of methods for solving Sudoku puzzles. Only methods that are differentiable are included in the comparison. Entries marked with an asterix are our own experiments, the rest are from the respective papers.

Method	Givens	Accuracy
*Recurrent Relational Network** (this work)	17	**96.6**%
Loopy BP, modified [Khan et al., 2014]	17	92.5%
Loopy BP, random [Bauke, 2008]	17	61.7%
Loopy BP, parallel [Bauke, 2008]	17	53.2%
Deeply Learned Messages* [Lin et al., 2015]	17	0%
Relational Network, node* [Santoro et al., 2017]	17	0%
Relational Network, graph* [Santoro et al., 2017]	17	0%
Deep Convolutional Network [Park, 2016]	24-36	70%

3.4 Age arithmetic

Anonymous reviewer 2 suggested the following task which we include here. The task is to infer the age of a person given a single absolute age and a set of age differences, e.g. "Alice is 20 years old. Alice is 4 years older than Bob. Charlie is 6 years younger than Bob. How old is Charlie?". Please see the supplementary material for details on the task and results.

4 Discussion

We have proposed a general relational reasoning model for solving tasks requiring an order of magnitude more complex relational reasoning than the current state-of-the art. BaBi and Sort-of-CLEVR require a few steps, Pretty-CLEVR requires up to eight steps and Sudoku requires more than ten steps. Our relational reasoning module can be added to any deep learning model to add a powerful relational reasoning capacity. We get state-of-the-art results on Sudokus solving 96.6% of the hardest Sudokus with 17 givens. We also markedly improve state-of-the-art on the BaBi dataset solving 20/20 tasks in 13 out of 15 runs with a single model trained jointly on all tasks.

One potential issue with having a loss at every step is that it might encourage the network to learn a greedy algorithm that gets stuck in a local minima. However, the output function r separates the node hidden states and messages from the output probability distributions. The network therefore has the capacity to use a small part of the hidden state for retaining a current best guess, which can remain constant over several steps, and other parts of the hidden state for running a non-greedy multi-step algorithm.

Sending messages for all nodes in parallel and summing all the incoming messages might seem like an unsophisticated approach that risk resulting in oscillatory behavior and drowning out the important messages. However, since the receiving node hidden state is an input to the message function, the receiving node can in a sense determine which messages it wishes to receive. As such, the sum can be seen as an implicit attention mechanism over the incoming messages. Similarly the network can learn an optimal message passing schedule, by ignoring messages based on the history and current state of the receiving and sending node.

5 Related work

Relational networks [Santoro et al., 2017] and interaction networks [Battaglia et al., 2016] are the most directly comparable to ours. These models correspond to using a single step of equation 3. Since it only does one step it cannot naturally do complex multi-step relational reasoning. In order to solve the tasks that require more than a single step it must compress all the relevant relations into a fixed size vector, then perform the remaining relational reasoning in the last forward layers. Relational networks, interaction networks and our proposed model can all be seen as an instance of Graph Neural Networks [Scarselli et al., 2009, Gilmer et al., 2017].

Graph neural networks with message passing computations go back to Scarselli et al. [2009]. However, there are key differences that we found important for implementing stable multi-step relational reasoning. Including the node features x_j at every step in eq. 3 is important to the stability of the network. Scarselli et al. [2009], eq. 3 has the node features, l_n, inside the message function. Battaglia et al. [2016] use an x_j in the node update function, but this is an external driving force. Sukhbaatar et al. [2016] also proposed to include the node features at every step. Optimizing the loss at every step in order to learn a convergent message passing algorithm is novel to the best of our knowledge. Scarselli et al. [2009] introduces an explicit loss term to ensure convergence. Ross et al. [2011] trains the inference machine predictors on every step, but there are no hidden states; the node states are the output marginals directly, similar to how belief propagation works.

Our model can also be seen as a completely learned message passing algorithm. Belief propagation is a hand-crafted message passing algorithm for performing exact inference in directed acyclic graphical models. If the graph has cycles, one can use a variant, loopy belief propagation, but it is not guaranteed to be exact, unbiased or converge. Empirically it works well though and it is widely used [Murphy et al., 1999]. Several works have proposed replacing parts of belief propagation with learned modules [Heess et al., 2013, Lin et al., 2015]. Our work differs by not being rooted in loopy BP, and instead learning all parts of a general message passing algorithm. Ross et al. [2011] proposes

Inference Machines which ditch the belief propagation algorithm altogether and instead train a series of regressors to output the correct marginals by passing messages on a graph. Wei et al. [2016] applies this idea to pose estimation using a series of convolutional layers and Deng et al. [2016] introduces a recurrent node update for the same domain.

There is rich literature on combining symbolic reasoning and logic with sub-symbolic distributed representations which goes all the way back to the birth of the idea of parallel distributed processing McCulloch and Pitts [1943]. See [Raedt et al., 2016, Besold et al., 2017] for two recent surveys. Here we describe only a few recent methods. Serafini and Garcez [2016] introduces the Logic Tensor Network (LTN) which describes a first order logic in which symbols are grounded as vector embeddings, and predicates and functions are grounded as tensor networks. The embeddings and tensor networks are then optimized jointly to maximize a fuzzy satisfiability measure over a set of known facts and fuzzy constraints. Šourek et al. [2015] introduces the Lifted Relational Network which combines relational logic with neural networks by creating neural networks from lifted rules and training examples, such that the connections between neurons created from the same lifted rules shares weights. Our approach differs fundamentally in that we do not aim to bridge symbolic and sub-symbolic methods. Instead we stay completely in the sub-symbolic realm. We do not introduce or consider any explicit logic, aim to discover (fuzzy) logic rules, or attempt to include prior knowledge in the form of logical constraints.

Amos and Kolter [2017] Introduces OptNet, a neural network layer that solve quadratic programs using an efficient differentiable solver. OptNet is trained to solve 4x4 Sudokus amongst other problems and beats the deep convolutional network baseline as described in Park [2016]. Unfortunately we cannot compare to OptNet directly as it has computational issues scaling to 9x9 Sudokus due to an implementation error (Brandon Amos, 2018, personal communication).

Sukhbaatar et al. [2016] proposes the Communication Network (CommNet) for learning multi-agent cooperation and communication using back-propagation. It is similar to our recurrent relational network, but differs in key aspects. The messages passed between all nodes at a given step are the same, corresponding to the average of all the node hidden states. Also, it is not trained to minimize the loss on every step of the algorithm.

Acknowledgments

We'd like to thank the anonymous reviewers for the valuable comments and suggestions, especially reviewer 2 who suggested the age arithmetic task. This research was supported by the NVIDIA Corporation with the donation of TITAN X GPUs.

References

Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. *arXiv preprint arXiv:1703.00443*, 2017.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

Heiko Bauke. Passing messages to lonely numbers. *Computing in Science & Engineering*, 10(2): 32–40, 2008.

Tarek R Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C Lamb, Daniel Lowd, Priscila Machado Vieira Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

Zhiwei Deng, Arash Vahdat, Hexiang Hu, and Greg Mori. Structure inference machines: Recurrent neural networks for analyzing relations in group activity recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4772–4781, 2016.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017.

Nicolas Heess, Daniel Tarlow, and John Winn. Learning to pass expectation propagation messages. In *Advances in Neural Information Processing Systems*, pages 3219–3227, 2013.

Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Sheehan Khan, Shahab Jabbari, Shahin Jabbari, and Majid Ghanbarinejad. Solving Sudoku using probabilistic graphical models. 2014.

Donald E Knuth. Dancing links. *arXiv preprint cs/0011047*, 2000.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 2016.

Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pages 361–369, 2015.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Peter Norvig. Solving every Sudoku puzzle, 2006. URL `http://norvig.com/sudoku.html`.

Kyubyong Park. Can neural networks crack Sudoku?, 2016. URL `https://github.com/Kyubyong/sudoku`.

Jack Rae, Jonathan J Hunt, Ivo Danihelka, Timothy Harley, Andrew W Senior, Gregory Wayne, Alex Graves, and Tim Lillicrap. Scaling memory-augmented neural networks with sparse reads and writes. In *Advances in Neural Information Processing Systems*, pages 3621–3629, 2016.

Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016.

Stephane Ross, Daniel Munoz, Martial Hebert, and J Andrew Bagnell. Learning message-passing inference machines for structured prediction. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2737–2744. IEEE, 2011.

Gordon Royle. Minimum Sudoku, 2014. URL `http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php`.

Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Luciano Serafini and Artur S d'Avila Garcez. Learning and reasoning with logic tensor networks. In *AI* IA 2016 Advances in Artificial Intelligence*, pages 334–348. Springer, 2016.

Gustav Šourek, Vojtech Aschenbrenner, Filip Železny, and Ondřej Kuželka. Lifted relational neural networks. In *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583*, pages 52–60. CEUR-WS. org, 2015.

Elizabeth S Spelke and Katherine D Kinzler. Core knowledge. *Developmental science*, 10(1):89–96, 2007.

Elizabeth S Spelke, Grant Gutheil, and Gretchen Van de Walle. The development of object perception. 1995.

Sainbayar Sukhbaatar, Jason Weston, Rob Fergus, et al. End-to-end memory networks. In *Advances in neural information processing systems*, pages 2440–2448, 2015.

Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M Rush, Bart van Merriënboer, Armand Joulin, and Tomas Mikolov. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.

Hyochang Yang, Sungzoon Cho, et al. Finding remo (related memory object): A simple neural architecture for text based reasoning. *arXiv preprint arXiv:1801.08459*, 2018.

6 Supplementary Material

6.1 bAbI experimental details

Unless otherwise specified we use 128 hidden units for all layers and all MLPs are 3 ReLU layers followed by a linear layer.

We compute each node feature vector as

$$x_i = \text{MLP}(\text{concat}(\text{last}(\text{LSTM}_S(s_i)), \text{last}(\text{LSTM}_Q(q)), \text{onehot}(p_i + o)))$$

where s_i is fact i, q is the question, p_i is the sentence position (1-20) of fact i and o is a random offset per question (1-20), such that the onehot output is 40 dimensional. The offset is constant for all facts related to a single question to avoid changing the relative order of the facts. The random offset prevents the network from memorizing the position of the facts and rather reason about their ordering. Our message function f is a MLP. Our node function g uses an LSTM over reasoning steps

$$h_j^t, s_j^t = \text{LSTM}_G(\text{MLP}(\text{concat}(x_j, m_j^t)), s_j^{t-1}) \ ,$$

where s_j^t is the cell state of the LSTM for unit j at time t. s_j^0 is initialized to zero.

We run our network for three steps. To get a graph level output, we use a MLP over the sum of the node hidden states, $o^t = \text{MLP}\left(\sum_i h_i^t\right)$ with 3 layers, the final being a linear layer that maps to the output logits. The last two layers uses dropout of 50%. We train and validate on all 20 tasks jointly using the 9,000 training and 1,000 validation samples defined in the `en_valid_10k` split. We use the Adam optimizer with a batch size of 512, a learning rate of 2e-4 and L2 regularization with a rate of 1e-5. We train for 5M gradient steps.

6.2 bAbI ablation experiments

To test which parts of the proposed model is important to solving the bAbI tasks we perform ablation experiments. One of the main differences between the relational network and our proposed model, aside from the recurrent steps, is that we encode the sentences and question together. We ablate the model in two ways to test how important this is. 1) Using a single linear layer instead of the 4-layer MLP baseline, and 2) Not encoding them together. In this case the node hidden states are initialized to the fact encodings. We found dropout to be important, so we also perform an ablation experiment without dropout. We run each ablation experiment eight times. We also do pseudo-ablation experiments with fewer steps by measuring at each step of the RRN. See table 3.

Model	Runs	Mean Error (%)	Failed tasks (err. >5%)	Mean error @ 1M updates (%)
Baseline, 3 steps	15	0.46 ± 0.77	0.13 ± 0.35	1.83 ± 1.06
Baseline, 2 steps	15	0.46 ± 0.76	0.13 ± 0.35	1.83 ± 1.06
Baseline, 1 step	15	0.48 ± 0.79	0.13 ± 0.35	1.84 ± 1.06
linear encoding	8	$\mathbf{0.20 \pm 0.01}$	$\mathbf{0 \pm 0}$	$\mathbf{0.63 \pm 0.69}$
no encoding	8	0.53 ± 0.91	0.13 ± 0.35	2.39 ± 1.73
no dropout	8	1.74 ± 1.28	0.63 ± 0.52	2.57 ± 0.95

Table 3: BaBi ablation results.

6.3 Pretty-CLEVR experimental details

Our setup for Pretty-CLEVR is a bit simpler than for bAbI. Unless otherwise specified we use 128 hidden units for all hidden layers and all MLPs are 1 ReLU layer followed by a linear layer.

We compute each node feature vector x_i as

$$\begin{aligned} o_i &= \text{concat}(p_i, \text{onehot}(c_i), \text{onehot}(m_i)) \\ q &= \text{concat}(\text{onehot}(s), \text{onehot}(n)) \\ x_i &= \text{MLP}(\text{concat}(o_i, q)) \end{aligned}$$

where $p_i \in [0, 1]^2$ is the position, $\mathbb{N}^n \equiv \{0, ..., n-1\}$, $c_i \in \mathbb{N}^8$ is the color, $m_i \in \mathbb{N}^8$ is the marker, $s \in \mathbb{N}^{16}$ is the marker or color of the start object, and $n \in \mathbb{N}^8$ is the number of jumps.

Our message function f is a MLP. Our node function g is,

$$h_j^t = \text{MLP}(\text{concat}(h_j^{t-1}, x_j, m_j^t))$$

Our output function r is a MLP with a dropout fraction of 0.5 in the penultimate layer. The last layer has 16 hidden linear units. We run our recurrent relational network for 4 steps.

We train on the 12.8M training questions, and augment the data by scaling and rotating the scenes randomly. We use separate validation and test sets of 128.000 questions each. We use the Adam optimizer with a learning rate of 1e-4 and train for 10M gradient updates with a batch size of 128.

The baseline RN is identical to the described RRN, except it only does a single step of relational reasoning.

The baseline MLP takes the entire scene state, $\mathbf{x}$, as an input, such that

$$\mathbf{x} = \text{concat}(o_0, ..., o_7, d_{00}, ..., d_{77}, q)$$

where $d_{ij} \in \mathbb{R}$ is the euclidean distance from object i to j.

The baseline MLP has 4 ReLu layers with 256 hidden units, with dropout of 0.5 on the last layer, followed by a linear layer with 16 hidden units. The baseline MLP has 87% more parameters than the RRN and RN (261,136 vs 139,536).

6.4 Sudoku dataset

To generate our dataset the starting point is the collection of 49,151 unique 17-givens puzzles gathered by Royle [2014] which we solve using the solver from Norvig [2006]. Then we split the puzzles into a test, validation and training *pool*, with 10,000, 1,000 and 38,151 samples respectively. To generate the *sets* we train, validate and test on we do the following: for each $n \in \{0, ..., 17\}$ we sample k puzzles from the respective pool, with replacement. For each sampled puzzle we add n random digits from the solution. We then swap the digits according to a random permutation, e.g. $1 \to 5$, $2 \to 3$, etc. The resulting puzzle is added to the respective set. For the test, validation and training sets we sample $k = 1,000$, $k = 1,000$ and $k = 10,000$ puzzles in this way.

6.5 Sudoku experimental details

Unless otherwise specified we use 96 hidden units for all hidden layers and all MLPs are 3 ReLU layers followed by a linear layer.

Denote the digit for cell j d_j (0-9, 0 if not given), and the row and column position row_j (1-9) and column_j (1-9) respectively.. The node features are then

$$x_j = \text{MLP}(\text{concat}(\text{embed}(d_j), \text{embed}(\text{row}_j), \text{embed}(\text{column}_j)))$$

where each embed is a 16 dimensional learned embedding. We could probably have used one-hot encoding instead of the embeddings, embedding was just the first thing we tried. Edge features were not used. The message function f is an MLP. The node function g, is identical to the setup for bAbI, i.e.

$$h_j^t, s_j^t = \text{LSTM}_G(\text{MLP}(\text{concat}(x_j, m_j^t)), s_j^{t-1}) \ .$$

The LSTM cell state is initialized to zeros.

The output function r is a linear layer with nine outputs to produce the output logits o_i^t. We run the network for 32 steps with a loss on every step. We train the model for 300.000 gradient updates with a batch size of 256 using Adam with a learning rate of 2e-4 and L2 regularization of 1e-4 on all weight matrices.

6.6 Sudoku relational network baseline details

The node centric corresponds exactly to a single step of our network. The graph centric approach is closer to the original relational network. It does one step of relational reasoning as our network, then sums all the node hidden states. The sum is then passed through a 4 layer MLP with $81 \cdot 9$ outputs, one for each cell and digit. The graph centric model has larger hidden states of 256 in all layers to compensate somewhat for the sum squashing the entire graph into a fixed size vector. Otherwise both networks are identical to our network. The graph centric has over 4 times as many parameters as our model (944,874 vs. 201,194) but performs worse than the node centric.

6.7 Age arithmetic task details

We generated all 262,144 unique trees with 8 nodes and split them 90%/10% into training and test graphs. The nodes represent the persons, and the edges which age differences will be given to the network. During training and testing we sample a batch of graphs from the respective set and sample 8 random ages (0-99) for each. We compute the absolute difference as well as the sign for each edge in the graphs. This gives us 7 relative facts on the form "person A (0-7), person B (0-7), younger/older (-1,1), absolute age difference (0-99)". Then we add the final fact which is the age of one of the nodes at random, e.g. "3, 3, 0, 47", using the zero sign to indicate this fact is absolute and not relative. The question is the age of one of the persons at random (0-7). For each graph we compute the shortest path from the anchor person to the person in question. This is the minimum number of arithmetic computations that must be performed to infer the persons age from the given facts.

The 8 facts (1 anchor, 7 relative) are given to the network as a fully connected graph of 8 nodes. Note, this graph is different from the tree used to generate the facts. The network never sees the tree. The input vector for each fact are the four fact integers and the question integer one-hot encoded and concatenated. We use the same architecture as for the bAbI experiments except all MLPs are 3 dense layers with 128 ReLu units followed by one linear layer. We train the network for 8 steps, and test it for each step. See figure 5 for results.

6.8 Unrolled recurrent relational network

6.9 Full Sudoku solution

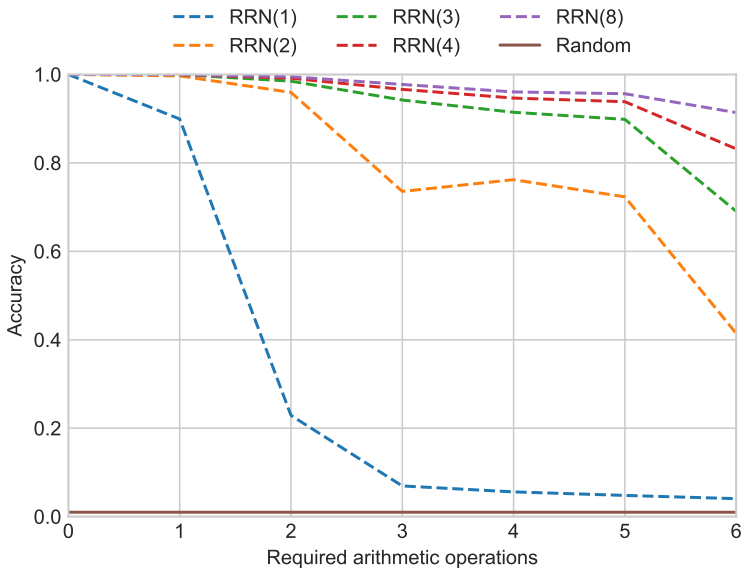
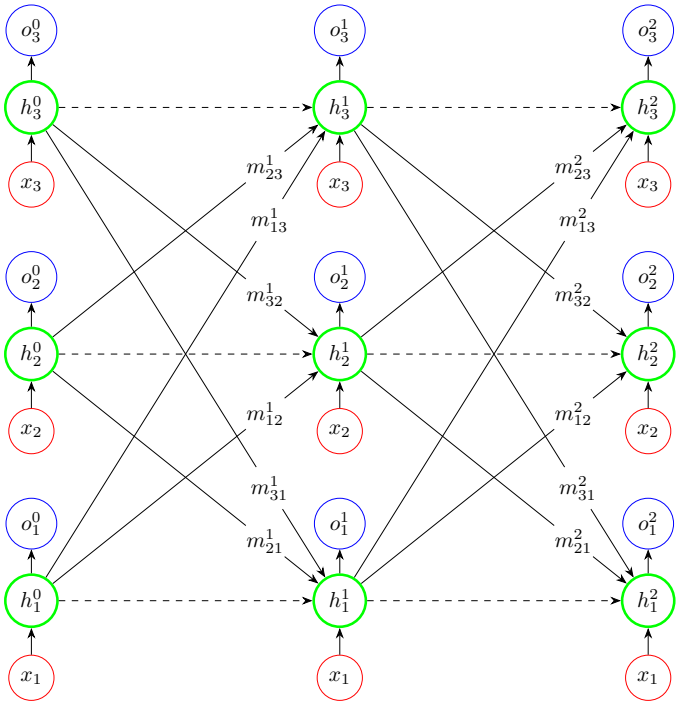


Figure 5: Results for the age arithmetic task. The number in parenthesis indicate how many steps the RRN was run during testing. Random corresponds to picking one of the 100 possible ages randomly.

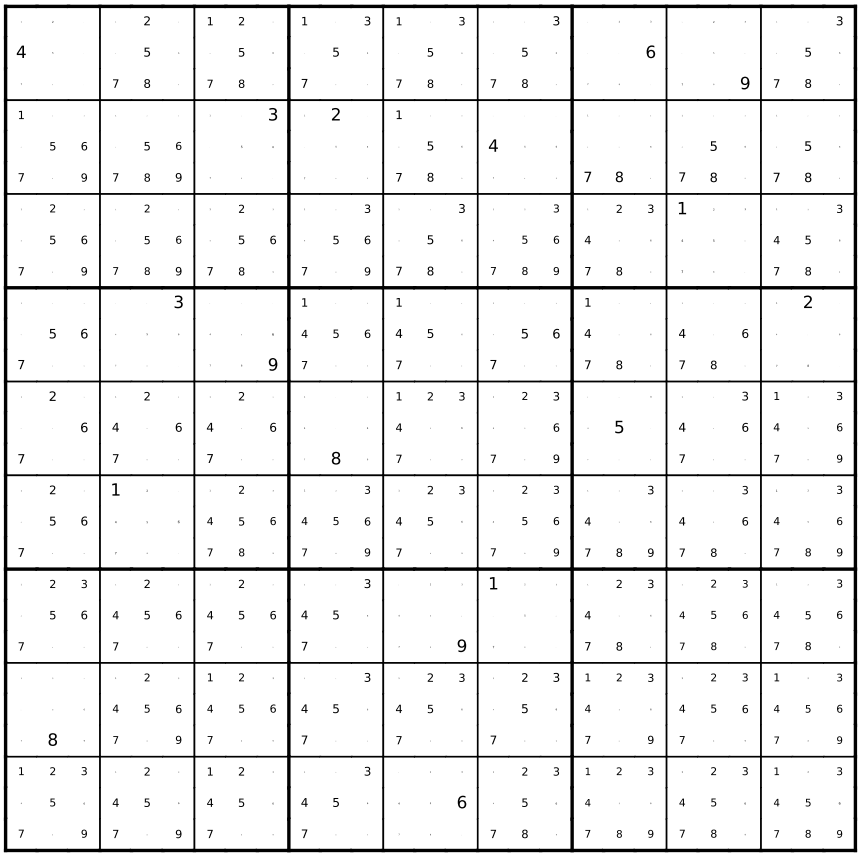


Recurrent relational network on a fully connected graph with 3 nodes. Subscripts denote node indices and superscripts denote steps t. The dashed lines indicate the recurrent connections.

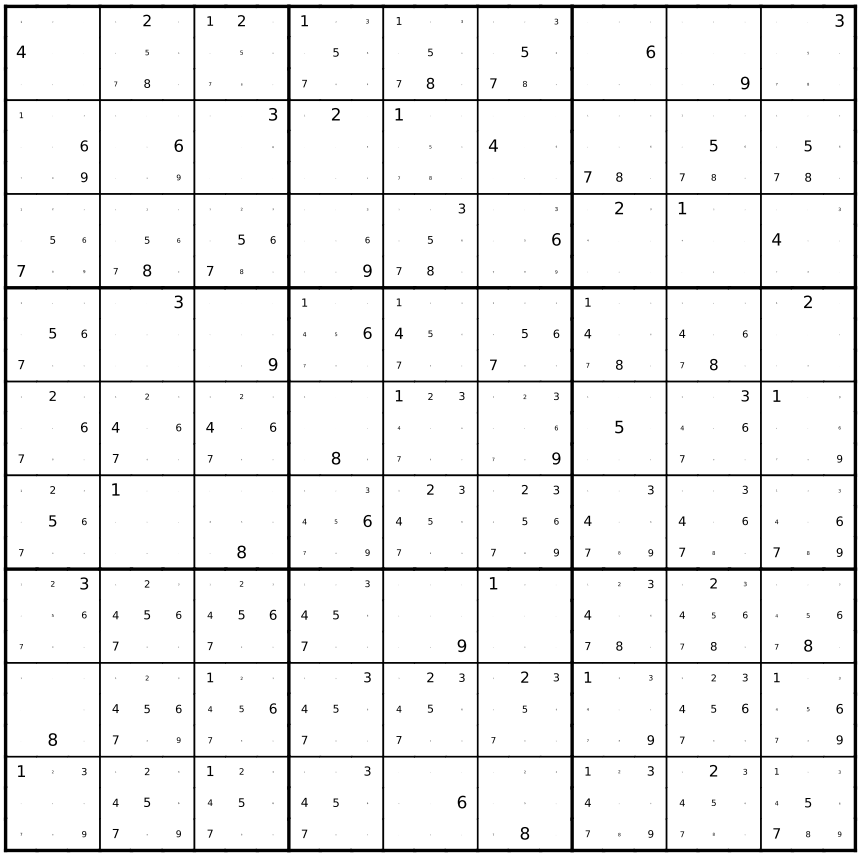
An example Sudoku. Each of the 81 cells contain each digit 1-9, which is useful if the reader wishes to try to solve the Sudoku as they can be crossed out or highlighted, etc. The digit font size corresponds to the probability our model assigns to each digit at step 0, i.e. before any steps are taken. Subsequent pages contains the Sudoku as it evolves with more steps of our model.

4															6				9	
					3		2					4								
																1				
				3													2			
					9															
													5							
							8													
			1																	
											1									
				8																
												6								

Step 1



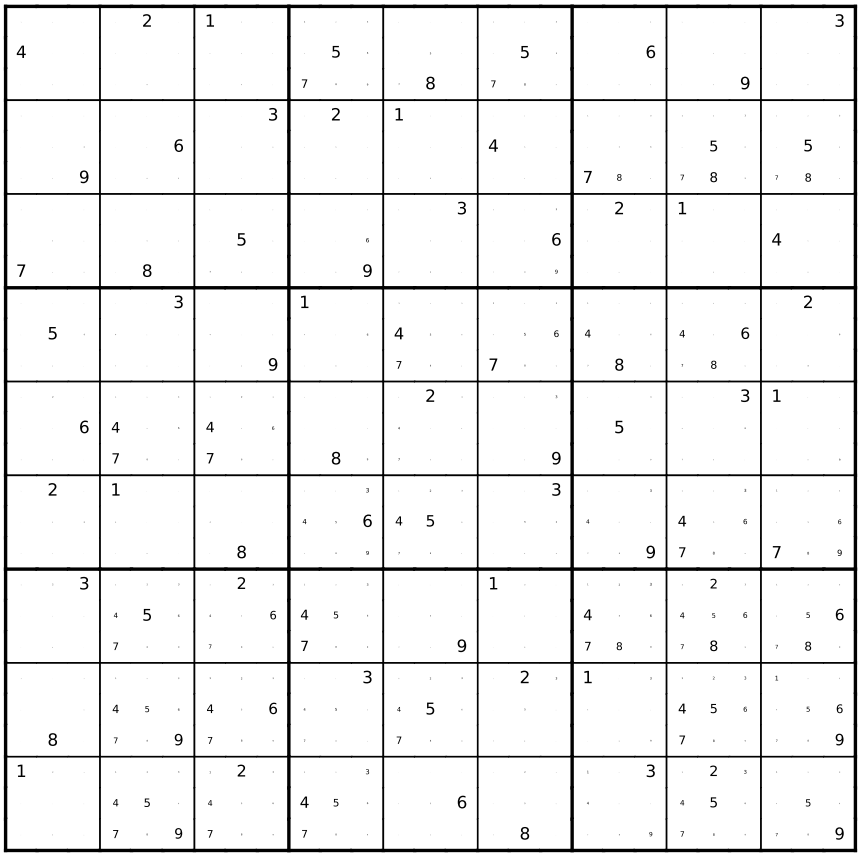
Step 4



Step 8

4	2	1	5 7	8	5 7	6	9	3
9	6	3	2	1	4	7 8	5 7 8	5 7 8
7	8	5	6 9	3	3 6 9	2	1	4
5 6	3	9	1	4 5 7	5 7	1 4 7 8	4 6 7 8	2
2 6	4 7	2 4 6 7	8	2 7	3 9	5	3 6	1
2 5	1	8	3 6 9	2 4 5 7	2 3 9	3 4 7 9	3 4 6 7	7 9
3	4 5 7	2 6 7	4 5 7	9	1	4 7 8	2 4 5 6 7 8	6 7 8
8	4 5 7 9	2 4 6 7	3 4 5 7	2 5 7	2 3	1 3 9	2 3 4 5 6 7	1 6 9
1	4 5 7 9	2 4 7	3 4 5 7	6	8	3 9	2 3 4 5 7	5 7 9

Step 12



Step 16

4	2	1	7	8	5	6	9	3
9	6	3	2	1	4	7	5 8	5 8
7	8	5	9	3	6	2	1	4
5	3	9	1	4	7	8	6	2
6	4 7	4 7	8	2	9	5	3	1
2	1	8	6	5	3	9	4	7
3	7	2 6 7	4 5	9	1	4	2 7 8	6 8
8	4 5 9	4 6	3	7	2	1	5	6 9
1	4 5 7 9	2 4 7	4 5	6	8	3	2 5 7	5 9

Step 20

4	2	1	7	8	5	6	9	3
9	6	3	2	1	4	7	8	5
7	8	5	9	3	6	2	1	4
5	3	9	1	4	7	8	6	2
6	4	7	8	2	9	5	3	1
2	1	8	6	5	3	9	4	7
3	7	6	5	9	1	4	2	8
8	9	4	3	7	2	1	5	6
1	5	2	4	6	8	3	7	9

Bibliography

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Heiko Bauke. Passing messages to lonely numbers. *Computing in Science & Engineering*, 10(2):32–40, 2008.

Mary Elaine Califf and Raymond J Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4(Jun):177–210, 2003.

Francesca Cesarini, Enrico Francesconi, Marco Gori, and Giovanni Soda. Analysis and understanding of multi-class invoices. *Document Analysis and Recognition*, 6 (2):102–114, 2003.

Laura Chiticariu, Yunyao Li, and Frederick R Reiss. Rule-based information extraction is dead! long live rule-based information extraction systems! In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 827–832, 2013.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Fabio Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'01, pages 1251–1256, 2001.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pages 2933–2941, 2014.

Andreas R Dengel and Bertin Klein. smartfix: A requirements-driven system for document analysis and understanding. In *International Workshop on Document Analysis Systems*, pages 433–444. Springer, 2002.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

George R Doddington, Alexis Mitchell, Mark A Przybocki, Lance A Ramshaw, Stephanie Strassel, and Ralph M Weischedel. The automatic content extraction (ace) program-tasks, data, and evaluation. In *LREC*, volume 2, page 1, 2004.

Daniel Esser, Daniel Schuster, Klemens Muthmann, Michael Berger, and Alexander Schill. Automatic indexing of scanned documents: a layout-based approach. In *Document Recognition and Retrieval XIX*, volume 8297, page 82970H. International Society for Optics and Photonics, 2012.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

Ralph Grishman and Beth Sundheim. Message understanding conference-6: A brief history. In *COLING 1996 Volume 1: The 16th International Conference on Computational Linguistics*, volume 1, 1996.

Sonal Gupta and Christopher Manning. Improved pattern learning for bootstrapped entity extraction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 98–108, 2014a.

Sonal Gupta and Christopher Manning. Spied: Stanford pattern based information extraction and diagnostics. In *Proceedings of the Workshop on Interactive Language Learning, Visualization, and Interfaces*, pages 38–44, 2014b.

Marti A. Hearst, Susan T Dumais, Edgar Osuna, John Platt, and Bernhard Scholkopf. Support vector machines. *IEEE Intelligent Systems and their applications*, 13(4): 18–28, 1998.

Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Scott B Huffman. Learning information extraction patterns from examples. In *International Joint Conference on Artificial Intelligence*, pages 246–260. Springer, 1995.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

Kenji Kawaguchi and Yoshua Bengio. Depth with nonlinearity creates no bad local minima in resnets. *arXiv preprint arXiv:1810.09038*, 2018.

Sheehan Khan, Shahab Jabbari, Shahin Jabbari, and Majid Ghanbarinejad. Solving Sudoku using probabilistic graphical models. unpublished, 2014.

Jun-Tae Kim and Dan I. Moldovan. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE transactions on knowledge and data engineering*, 7(5):713–724, 1995.

Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86, 2005.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436, 2015.

Maria Liakata, Larisa N Soldatova, et al. Semantic annotation of papers: Interface & enrichment tool (sapient). In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pages 193–200. Association for Computational Linguistics, 2009.

Guosheng Lin, Chunhua Shen, Ian Reid, and Anton van den Hengel. Deeply learning the messages in message passing inference. In *Advances in Neural Information Processing Systems*, pages 361–369, 2015.

Eric Medvet, Alberto Bartoli, and Giorgio Davanzo. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(4):335–347, 2011.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labeled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

Ion Muslea et al. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 workshop on machine learning for information extraction*, volume 2. Orlando Florida, 1999.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Peter Norvig. Solving every Sudoku puzzle. http://norvig.com/sudoku.html viewed on 2017-10-12, 2006.

Rasmus Berg Palm, Dirk Hovy, Florian Laws, and Ole Winther. End-to-end information extraction without token-level supervision. *arXiv preprint arXiv:1707.04913*, 2017a.

Rasmus Berg Palm, Ole Winther, and Florian Laws. Cloudscan - a configuration-free invoice analysis system using recurrent neural networks. *arXiv preprint arXiv:1708.07403*, 2017b.

Rasmus Berg Palm, Florian Laws, and Ole Winther. Attend, copy, parse - end-to-end information extraction from documents. unpublished, 2018a.

Rasmus Berg Palm, Ulrich Paquet, and Ole Winther. Recurrent relational networks. In *Advances in neural information processing systems*, 2018b.

Kyubyong Park. Can neural networks crack Sudoku? https://github.com/Kyubyong/sudoku viewed on 2017-10-17, 2016.

Siddharth Patwardhan. *Widening the field of view of information extraction through sentential event recognition*. PhD thesis, Citeseer, 2010.

Patti J Price. Evaluation of spoken language systems: The atis domain. In *Speech and Natural Language: Proceedings of a Workshop Held at Hidden Valley, Pennsylvania, June 24-27, 1990*, 1990.

Lance Ramshaw and Mitch Marcus. Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*, 1995.

Ellen Riloff et al. Automatically constructing a dictionary for information extraction tasks. In *AAAI*, volume 1, pages 2–1. Citeseer, 1993.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Marçal Rusinol, Tayeb Benkhelfallah, and Vincent Poulain dAndecy. Field extraction from administrative documents by incremental structural templates. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1100–1104. IEEE, 2013.

Adam Santoro, David Raposo, David GT Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. *arXiv preprint arXiv:1706.01427*, 2017.

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

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

Daniel Schuster, Klemens Muthmann, Daniel Esser, Alexander Schill, Michael Berger, Christoph Weidling, Kamil Aliyev, and Andreas Hofmeier. Intellix–end-user trained information extraction for document archiving. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 101–105. IEEE, 2013.

Stephen Soderland, David Fisher, Jonathan Aseltine, and Wendy Lehnert. Crystal: Inducing a conceptual dictionary. *arXiv preprint cmp-lg/9505020*, 1995.

Beth M Sundheim. Overview of the third message understanding evaluation and conference. In *Proceedings of the 3rd conference on Message understanding*, pages 3–16. Association for Computational Linguistics, 1991.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

Alan Turing. Computing machinery and intelligence. *Mind*, 59(236):433, 1950.

Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, page 125, 2016.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Roman V Yampolskiy. Turing test as a defining feature of ai-completeness. In *Artificial intelligence, evolutionary computing and metaheuristics*, pages 3–17. Springer, 2013.

Xiao Yang, Ersin Yumer, Paul Asente, Mike Kraley, Daniel Kifer, and C Lee Giles. Learning to extract semantic structure from documents using multimodal fully convolutional neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*, 2014.