Making Neural QA as Simple as Possible but not Simpler

Dirk Weissenborn **Georg Wiese** **Laura Seiffe**
Language Technology Lab, DFKI
Alt-Moabit 91c
Berlin, Germany
{dirk.weissenborn, georg.wiese, laura.seiffe}@dfki.de

Abstract

Recent development of large-scale question answering (QA) datasets triggered a substantial amount of research into end-to-end neural architectures for QA. Increasingly complex systems have been conceived without comparison to simpler neural baseline systems that would justify their complexity. In this work, we propose a simple heuristic that guides the development of neural baseline systems for the extractive QA task. We find that there are two ingredients necessary for building a high-performing neural QA system: first, the awareness of question words while processing the context and second, a composition function that goes beyond simple bag-of-words modeling, such as recurrent neural networks. Our results show that FastQA, a system that meets these two requirements, can achieve very competitive performance compared with existing models. We argue that this surprising finding puts results of previous systems and the complexity of recent QA datasets into perspective.

1 Introduction

Question answering is an important end-user task at the intersection of natural language processing (NLP) and information retrieval (IR). QA systems can bridge the gap between IR-based search engines and sophisticated intelligent assistants that enable a more directed information retrieval process. Such systems aim at finding precisely the piece of information sought by the user instead of documents or snippets containing the answer. A special form of QA, namely extractive QA, deals with the extraction of a direct *answer* to a *question* from a given textual *context*.

The creation of large-scale, extractive QA datasets (Rajpurkar et al., 2016; Trischler et al., 2017; Nguyen et al., 2016) sparked research interest into the development of end-to-end neural QA systems. A typical neural architecture consists of an embedding-, encoding-, interaction- and answer layer (Wang and Jiang, 2017; Yu et al., 2017; Xiong et al., 2017; Seo et al., 2017; Yang et al., 2017; Wang et al., 2017). Most such systems describe several innovations for the different layers of the architecture with a special focus on developing powerful *interaction layer* that aims at modeling word-by-word interaction between question and context.

Although a variety of extractive QA systems have been proposed, there is no competitive neural baseline. Most systems were built in what we call a *top-down* process that proposes a complex architecture and validates design decisions by an ablation study. Most ablation studies, however, remove only a single part of an overall complex architecture and therefore lack comparison to a reasonable neural baseline. This gap raises the question whether the complexity of current systems is justified solely by their empirical results.

Another important observation is the fact that seemingly complex questions might be answerable by simple heuristics. Let's consider the following example:

> ***When** did building activity occur on St. Kazimierz Church?*
>
> Building activity occurred in numerous noble palaces and churches [...]. One of the best examples [..] are Krasinski Palace (1677-1683), Wilanow Palace (1677-1696) and St. Kazimierz Church (**1688-1692**)

Although it seems that evidence synthesis of multiple sentences is necessary to fully understand the

Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017), pages 271–280,
Vancouver, Canada, August 3 - August 4, 2017.

relation between the answer and the question, answering this question is easily possible by applying a simple *context/type matching heuristic*. The heuristic aims at selecting answer spans that a) match the expected answer type (a time as indicated by "When") and b) are close to important question words ("St. Kazimierz Church"). The actual answer "1688-1692" would easily be extracted by such a heuristic.

In this work, we propose to use the aforementioned *context/type matching heuristic* as a guideline to derive simple neural baseline architectures for the extractive QA task. In particular, we develop a simple neural, bag-of-words (BoW)- and a recurrent neural network (RNN) baseline, namely *FastQA*. Crucially, both models do not make use of a complex interaction layer but model interaction between question and context only through computable features on the word level. FastQA's strong performance questions the necessity of additional complexity, especially in the interaction layer, which is exhibited by recently developed models. We address this question by evaluating the impact of extending FastQA with an additional interaction layer (FastQAExt) and find that it doesn't lead to systematic improvements. Finally, our contributions are the following: **i)** definition and evaluation of a BoW- and RNN-based neural QA baselines guided by a simple heuristic; **ii)** bottom-up evaluation of our FastQA system with increasing architectural complexity, revealing that the awareness of question words and the application of a RNN are enough to reach state-of-the-art results; **iii)** a complexity comparison between FastQA and more complex architectures as well as an in-depth discussion of usefulness of an interaction layer; **iv)** a qualitative analysis indicating that FastQA mostly follows our heuristic which thus constitutes a strong baseline for extractive QA.

2 A Bag-of-Words Neural QA System

We begin by motivating our architectures by defining our proposed context/type matching heuristic: a) the type of the answer span should correspond to the expected answer type given by the question, and b) the correct answer should further be surrounded by a context that fits the question, or, more precisely, it should be surrounded by many question words. Similar heuristics were frequently implemented explicitly in traditional QA systems, e.g., in the answer extraction step of Moldovan et al. (1999), however, in this work our heuristic is merely used as a guideline for the construction of neural QA systems. In the following, we denote the hidden dimensionality of the model by n, the question tokens by $Q = (q_1, ..., q_{L_Q})$, and the context tokens by $X = (x_1, ..., x_{L_X})$.

2.1 Embedding

The embedding layer is responsible for mapping tokens x to their corresponding n-dimensional representation $\boldsymbol{x}$. Typically this is done by mapping each word x to its corresponding word embedding $\boldsymbol{x}^w$ (*lookup-embedding*) using an embedding matrix E, s.t. $\boldsymbol{x}^w = Ex$. Another approach is to embed each word by encoding their corresponding character sequence $x^c = (c_1, ..., c_{L_X})$ with C, s.t. $\boldsymbol{x}^c = C(x_c)$ (*char-embedding*). In this work, we use a convolutional neural network for C of filter width 5 with max-pooling over time as explored by Seo et al. (2017), to which we refer the reader for additional details. Both approaches are combined via concatenation, s.t. the final embedding becomes $\boldsymbol{x} = [\boldsymbol{x}^w; \boldsymbol{x}^c] \in \mathbb{R}^d$.

2.2 Type Matching

For the BoW baseline, we extract the span in the question that refers to the expected, lexical answer type (LAT) by extracting either the question word(s) (e.g., *who, when, why, how, how many*, etc.) or the first noun phrase of the question after the question words "what" or "which" (e.g., "what *year* did..."). [1] This leads to a correct LAT for most questions. We encode the LAT by concatenating the embedding of the first- and last word together with the average embedding of all words within the LAT. The concatenated representations are further transformed by a fully-connected layer followed by a $\tanh$ non-linearity into $\tilde{\boldsymbol{z}} \in \mathbb{R}^n$. Note that we refer to a fully-connected layer in the following by FC, s.t. $\mathrm{FC}(\boldsymbol{u}) = W\boldsymbol{u} + \boldsymbol{b}$, $W \in \mathbb{R}^{n \times m}, \boldsymbol{b} \in \mathbb{R}^n, \boldsymbol{u} \in \mathbb{R}^m$.

We similarly encode each potential answer span (s, e) in the context, i.e., all spans with a specified, maximum number of words (10 in this work), by concatenating the embedding of the first- and last word together with the average embedding of all words within the span. Because the surrounding context of a potential answer span can give important clues towards the type of an answer span,

[1] More complex heuristics can be employed here but for the sake of simplicity we chose a very simple approach.

for instance, through nominal modifiers left of the span (e.g., "... president obama ...") or through an apposition right of the span (e.g., "... obama, president of..."), we additionally concatenate the average embeddings of the 5 words to the left and to the right of a span, respectively. The concatenated span representation, which comprises in total five different embeddings, is further transformed by a fully-connected layer with a $\tanh$ non-linearity into $\tilde{\boldsymbol{x}}_{s,e} \in \mathbb{R}^n$.

Finally, the concatenation of the LAT representation, the span representation and their element-wise product, i.e., $[\tilde{\boldsymbol{z}}; \tilde{\boldsymbol{x}}_{s,e}; \tilde{\boldsymbol{z}} \odot \tilde{\boldsymbol{x}}_{s,e}]$, serve as input to a feed-forward neural network with one hidden layer which computes the type score $g_{type}(s, e)$ for each span (s, e).

2.3 Context Matching

In order to account for the number of surrounding words of an answer span as a measure for question to answer span match (context match), we introduce two word-in-question features. They are computed for each context word x_j and explained in the following

binary The binary word-in-question (wiq^b) feature is 1 for tokens that are part of the question and else 0. The following equation formally defines this feature where $\mathbb{I}$ denotes the indicator function:

$$\text{wiq}^b_j = \mathbb{I}(\exists i : x_j = q_i) \tag{1}$$

weighted The wiq^w_j feature for context word x_j is defined in Eq. 3, where Eq. 2 defines a basic similarity score between q_i and x_j based on their word-embeddings. It is motivated on the one hand by the intuition that question tokens which rarely appear in the context are more likely to be important for answering the question, and on the other hand by the fact that question words might occur as morphological variants, synonyms or related words in the context. The latter can be captured (softly) by using word embeddings instead of the words themselves whereas the former is captured by the application of the softmax operation in Eq. 3 which ensures that infrequent occurrences of words are weighted more heavily.

$$sim_{i,j} = \boldsymbol{v}_{wiq}(\boldsymbol{x}_j \odot \boldsymbol{q}_i) \quad , \boldsymbol{v}_{wiq} \in \mathbb{R}^n \tag{2}$$

$$\text{wiq}^w_j = \sum_i \text{softmax}(sim_{i,.})_j \tag{3}$$

A derivation that connects wiq^w with the term-frequencies (a prominent information retrieval measure) of a word in the question and the context, respectively, is provided in Appendix A.

Finally, for each answer span (s, e) we compute the average wiq^b and wiq^w scores of the 5, 10 and 20 token-windows to the left and to the right of the respective (s, e)-span. This results in a total of 2 (kinds of features)×3 (windows)×2 (left/right) = 12 scores which are weighted by trainable scalar parameters and summed to compute the context-matching score $g_{ctxt}(s, e)$.

2.4 Answer Span Scoring

The final score g for each span (s, e) is the sum of the type- and the context matching score: $g(s, e) = g_{type}(s, e) + g_{ctxt}(s, e)$. The model is trained to minimize the softmax-cross-entropy loss given the scores for all spans.

3 FastQA

Although our BoW baseline closely models our intended heuristic, it has several shortcomings. First of all, it cannot capture the compositionality of language making the detection of sensible answer spans harder. Furthermore, the semantics of a question is dramatically reduced to a BoW representation of its expected answer-type and the scalar word-in-question features. Finally, answer spans are restricted to a certain length.

To account for these shortcomings we introduce another baseline which relies on the application of a single bi-directional recurrent neural networks (BiRNN) followed by a answer layer that separates the prediction of the start and end of the answer span. Lample et al. (2016) demonstrated that BiRNNs are powerful at recognizing named entities which makes them sensible choice for context encoding to allow for improved type matching. Context matching can similarly be achieved with a BiRNN by informing it of the locations of question tokens appearing in the context through our wiq-features. It is important to recognize that our model should implicitly learn to capture the heuristic, but is not limited by it.

On an abstract level, our RNN-based model, called FastQA, consists of three basic layers, namely the embedding-, encoding- and answer layer. Embeddings are computed as explained in §2.1. The other two layers are described in detail in the following. An illustration of the basic archi-

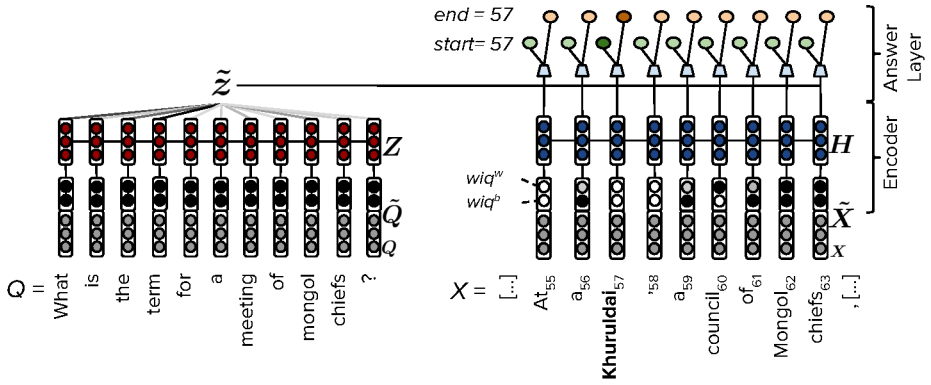

Figure 1: Illustration of FastQA system on example question from SQuAD. The two word-in-question features (wiq^b, wiq^w) are presented with varying degrees of activation.

tecture is provided in Figure 1.

3.1 Encoding

In the following, we describe the encoding of the context which is analogous to that of the question.

To allow for interaction between the two embeddings described in §2.1, they are first projected jointly to a n-dimensional representation (Eq. 4)) and further transformed by a single highway layer (Eq. 5) similar to Seo et al. (2017).

$$\boldsymbol{x}' = P\boldsymbol{x} \quad , P \in \mathbb{R}^{n \times d} \tag{4}$$

$$\boldsymbol{g}_e = \boldsymbol{\sigma}(\text{FC}(\boldsymbol{x}')) \, , \, \boldsymbol{x}'' = \tanh\left(\text{FC}\left(\boldsymbol{x}'\right)\right)$$
$$\tilde{\boldsymbol{x}} = \boldsymbol{g}_e\boldsymbol{x}' + (1 - \boldsymbol{g}_e)\boldsymbol{x}'' \tag{5}$$

Because we want the encoder to be aware of the question words we feed the binary- and the weighted *word-in-question* feature of §2.3 in addition to the embedded context words as input. The complete input $\tilde{\boldsymbol{X}} \in \mathbb{R}^{n+2\times L_X}$ to the encoder is therefore defined as follows:

$$\tilde{\boldsymbol{X}} = ([\tilde{\boldsymbol{x}}_1; \text{wiq}_1^b; \text{wiq}_1^w], ..., [\tilde{\boldsymbol{x}}_{L_X}; \text{wiq}_{L_X}^b; \text{wiq}_{L_X}^w])$$

$\tilde{\boldsymbol{X}}$ is fed to a bidirectional RNN and its output is again projected to allow for interaction between the features accumulated in the forward and backward RNN (Eq. 6). In preliminary experiments we found LSTMs (Hochreiter and Schmidhuber, 1997) to perform best.

$$\boldsymbol{H'} = \text{Bi-LSTM}(\tilde{\boldsymbol{X}}) \quad , \boldsymbol{H'} \in \mathbb{R}^{2n \times L_X}$$
$$\boldsymbol{H} = \tanh(B\boldsymbol{H'}^\top) \quad , B \in \mathbb{R}^{n \times 2n} \tag{6}$$

We initialize the projection matrix B with $[I_n; I_n]$, where I_n denotes the n-dimensional identity matrix. It follows that $\boldsymbol{H}$ is the sum of the outputs from the forward- and backward-LSTM at the beginning of training.

As mentioned before, we utilize the same encoder parameters for both question and context, except the projection matrix B which is not shared. However, they are initialized the same way, s.t. the context and question encoding is identical at the beginning of training. Finally, to be able to use the same encoder for both question and context we fix the two wiq features to 1 for the question.

3.2 Answer Layer

After encoding context X to $\boldsymbol{H} = [\boldsymbol{h}_1, ..., \boldsymbol{h}_{L_X}]$ and the question Q to $\boldsymbol{Z} = [\boldsymbol{z}_1, ..., \boldsymbol{z}_{L_Q}]$, we first compute a weighted, n-dimensional question representation $\tilde{\boldsymbol{z}}$ of Q (Eq. 7). Note that this representation is context-independent and as such only computed once, i.e., there is no additional word-by-word interaction between context and question.

$$\alpha = \text{softmax}(\boldsymbol{v}_q\boldsymbol{Z}) \quad , \boldsymbol{v}_q \in \mathbb{R}^n$$
$$\tilde{\boldsymbol{z}} = \sum_i \alpha_i \boldsymbol{z}_i \tag{7}$$

The probability distribution p_s for the start location of the answer is computed by a 2-layer feed-forward neural network with a rectified-linear (ReLU) activated, hidden layer $\boldsymbol{s}_j$ as follows:

$$\boldsymbol{s}_j = \text{ReLU}\left(\text{FC}\left([\boldsymbol{h}_j; \tilde{\boldsymbol{z}}; \boldsymbol{h}_j \odot \tilde{\boldsymbol{z}}]\right)\right)$$
$$p_s(j) \propto \exp(\boldsymbol{v}_s\boldsymbol{s}_j) \quad , \boldsymbol{v}_s \in \mathbb{R}^n \tag{8}$$

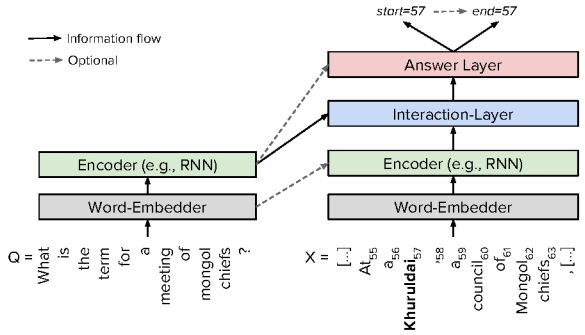

Figure 2: Illustration of the basic architecture which underlies most existing neural QA systems.

The conditional probability distribution p_e for the end location conditioned on the start location s is computed similarly by a feed-forward neural network with hidden layer $\boldsymbol{e}_j$ as follows:

$$\boldsymbol{e}_j = \text{ReLU}\left(\text{FC}\left([\boldsymbol{h}_j; \boldsymbol{h}_s; \tilde{\boldsymbol{z}}; \boldsymbol{h}_j \odot \tilde{\boldsymbol{z}}; \boldsymbol{h}_j \odot \boldsymbol{h}_s]\right)\right)$$
$$p_e(j|s) \propto \exp(\boldsymbol{v}_e \boldsymbol{e}_j) \quad , \boldsymbol{v}_e \in \mathbb{R}^n \tag{9}$$

The overall probability p of predicting an answer span (s, e) is $p(s, e) = p_s(s) \cdot p_e(e|s)$. The model is trained to minimize the cross-entropy loss of the predicted span probability $p(s, e)$.

Beam-search During prediction time, we compute the answer span with the highest probability by employing beam-search using a beam-size of k. This means that ends for the top-k starts are predicted and the span with the highest overall probability is predicted as final answer.

4 Comparison to Prior Architectures

Many neural architectures for extractive QA have been conceived very recently. Most of these systems can be broken down into four basic layers for which individual innovations were proposed. A high-level illustration of these systems is show in Figure 2. In the following, we compare our system in more detail with existing models.

Embedder The embedder is responsible for embedding a sequence of tokens into a sequence of n-dimensional states. Our proposed embedder (§2.1) is very similar to existing ones used for example in Seo et al. (2017); Yang et al. (2017).

Encoder Embedded tokens are further encoded by some form of composition function. A prominent type of encoder is the (bi-directional) recurrent neural network (RNN) which is also used in this work. Feeding additional word-level features similar to ours is rarely done with the exception of Wang et al. (2017); Li et al. (2016).

Interaction Layer Most research focused on the interaction layer which is responsible for word-by-word interaction between context and question. Different ideas were explored such as attention (Wang and Jiang, 2017; Yu et al., 2017), co-attention (Xiong et al., 2017), bi-directional attention flow (Seo et al., 2017), multi-perspective context matching (Wang et al., 2017) or fine-grained gating (Yang et al., 2017). All of these ideas aim at enriching the encoded context with weighted states from the question and in some cases also from the context. These are gathered individually for each context state, concatenated with it and serve as input to an additional RNN. Note that this layer is omitted completely in FastQA and therefore constitutes the main simplification over previous work.

Answer Layer Finally, most systems divide the prediction the start and the end by another network. Their complexity ranges from using a single fully-connected layer (Seo et al., 2017; Wang et al., 2017) to employing convolutional neural networks (Trischler et al., 2017) or recurrent, deep Highway-Maxout-Networks(Xiong et al., 2017). We further introduce *beam-search* to extract the most likely answer span with a simple 2-layer feed-forward network.

5 FastQA Extended

To explore the necessity of the interaction layer and to be architecturally comparable to existing models we extend FastQA with an additional interaction layer (FastQAExt). In particular, we introduce *representation fusion* to enable the exchange of information in between passages of the context (*intra-fusion*), and between the question and the context (*inter-fusion*). Representation fusion is defined as the weighted addition between a state, i.e., its n-dimensional representation, and its respective co-representation. For each context state its corresponding co-representation is retrieved via attention from the rest of the context (intra) or the question (inter), respectively, and "fused" into

its own representation. For the sake of brevity we describe technical details of this layer in Appendix B, because this extension is not the focus of this work but merely serves as a representative of the more complex architectures described in §4.

6 Experimental Setup

We conduct experiments on the following datasets.

SQuAD The Stanford Question Answering Dataset (Rajpurkar et al., 2016)[2] comprises over $100k$ questions about paragraphs of 536 Wikipedia articles.

NewsQA The NewsQA dataset (Trischler et al., 2017)[3] contains $100k$ answerable questions from a total of $120k$ questions. The dataset is built from CNN news stories that were originally collected by Hermann et al. (2015).

Performance on the SQuAD and NewsQA datasets is measured in terms of *exact match* (accuracy) and a mean, per answer token-based *F1* measure which was originally proposed by Rajpurkar et al. (2016) to also account for partial matches.

6.1 Implementation Details

BoW Model The BoW model is trained on spans up to length 10 to keep the computation tractable. This leads to an upper bound of about 95% accuracy on SQuAD and 87% on NewsQA. As pre-processing steps we lowercase all inputs and tokenize it using spacy[4]. The binary word in question feature is computed on lemmas provided by spacy and restricted to alphanumeric words that are not stopwords. Throughout all experiments we use a hidden dimensionality of $n = 150$, dropout at the input embeddings with the same mask for all words (Gal and Ghahramani, 2015) and a rate of 0.2 and 300-dimensional fixed word-embeddings from `Glove` (Pennington et al., 2014). We employed ADAM (Kingma and Ba, 2015) for optimization with an initial learning-rate of 10^{-3} which was halved whenever the $F1$ measure on the development set dropped between epochs. We used mini-batches of size 32.

FastQA The pre-processing of FastQA is slightly simpler than that of the BoW model. We tokenize the input on whitespaces (exclusive) and non-alphanumeric characters (inclusive). The binary word in question feature is computed on the words as they appear in context. Throughout all experiments we use a hidden dimensionality of $n = 300$, variational dropout at the input embeddings with the same mask for all words (Gal and Ghahramani, 2015) and a rate of 0.5 and 300-dimensional fixed word-embeddings from `Glove` (Pennington et al., 2014). We employed ADAM (Kingma and Ba, 2015) for optimization with an initial learning-rate of 10^{-3} which was halved whenever the $F1$ measure on the development set dropped between checkpoints. Checkpoints occurred after every 1000 mini-batches each containing 64 examples.

Cutting Context Length Because NewsQA contains examples with very large contexts (up to more than 1500 tokens) we cut contexts larger than 400 tokens in order to efficiently train our models. We ensure that at least one, but at best all answers are still present in the remaining 400 tokens. Note that this restriction is only employed during training.

7 Results

7.1 Model Component Analysis

Model	Dev	
	F1	Exact
Logistic Regression[1]	51.0	40.0
Neural BoW Baseline	56.2	43.8
BiLSTM	58.2	48.7
BiLSTM + wiq^b	71.8	62.3
BiLSTM + wiq^w	73.8	64.3
BiLSTM + wiq^{b+w} (FastQA*)	74.9	65.5
FastQA* + intrafusion	76.2	67.2
FastQA* + intra + inter (FastQAExt*)	77.5	68.4
FastQA* + char-emb. (FastQA)	76.3	67.6
FastQAExt* + char-emb. (FastQAExt)	78.3	69.9
FastQA w/ beam-size 5	76.3	67.8
FastQAExt w/ beam-size 5	**78.5**	**70.3**

Table 1: SQuAD results on development set for increasingly complex architectures. [1]Rajpurkar et al. (2016)

Table 1 shows the individual contributions of each model component that was incrementally added to a plain BiLSTM model without features, character embeddings and beam-search. We see that the most crucial performance boost stems

[2] https://rajpurkar.github.io/SQuAD-explorer/
[3] https://datasets.maluuba.com/NewsQA/
[4] http://spacy.io

from the introduction of either one of our features ($\approx$ 15% F1). However, all other extensions also achieve notable improvements typically between 1 and 2% F1. Beam-search slightly improves results which shows that the most probable start is not necessarily the start of the best answer span.

In general, these results are interesting in many ways. For instance, it is surprising that a simple binary feature like wiq^b can have such a dramatic effect on the overall performance. We believe that the reason for this is the fact that an encoder without any knowledge of the actual question has to account for every possible question that might be asked, i.e., it has to keep track of the entire context around each token in its recurrent state. An informed encoder, on the other hand, can selectively keep track of question related information. It can further abstract over concrete entities to their respective types because it is rarely the case that many entities of the same type occur in the question. For example, if a person is mentioned in the question the context encoder only needs to remember that the "question-person" was mentioned but not the concrete name of the person.

Another interesting finding is the fact that additional character based embeddings have a notable effect on the overall performance which was already observed by Seo et al. (2017); Yu et al. (2017). We see further improvements when employing representation fusion to allow for more interaction. This shows that a more sophisticated interaction layer can help. However, the differences are not substantial, indicating that this extension does not offer any systematic advantage.

7.2 Comparing to State-of-the-Art

Our neural BoW baseline achieves good results on both datasets (Tables 3 and 1)[5]. For instance, it outperforms a feature rich logistic-regression baseline on the SQuAD development set (Table 1) and nearly reaches the BiLSTM baseline system (i.e., FastQA without character embeddings and features). It shows that more than half or more than a third of all questions in SQuAD or NewsQA, respectively, are (partially) answerable by a very simple neural BoW baseline. However, the gap to state-of-the-art systems is quite large ($\approx$ 20%F1) which indicates that employing more complex composition functions than averaging, such as RNNs in FastQA, are indeed necessary to achieve good performance.

[5]We did not evaluate the BoW baseline on the SQuAD test set because it requires submitting the model to Rajpurkar et al. (2016) and we find that comparisons on NewsQA and the SQuAD development set give us enough insights.

Model	**Test**	
	F1	Exact
Logistic Regression[1]	51.0	40.4
Match-LSTM[2]	73.7	64.7
Dynamic Chunk Reader[3]	71.0	62.5
Fine-grained Gating[4]	73.3	62.5
Multi-Perspective Matching[5]	75.1	65.5
Dynamic Coattention Networks[6]	75.9	66.2
Bidirectional Attention Flow[7]	77.3	68.0
r-net[8]	77.9	69.5
FastQA w/ beam-size $k = 5$	77.1	68.4
FastQAExt $k = 5$	**78.9**	**70.8**

Table 2: Official SQuAD leaderboard of single-model systems on test set from 2016/12/29, the date of submitting our model. [1]Rajpurkar et al. (2016), [2]Wang and Jiang (2017), [3]Yu et al. (2017), [4]Yang et al. (2017), [5]Wang et al. (2017), [6]Xiong et al. (2017), [7]Seo et al. (2017), [8] not published. Note that systems are regularly uploaded and improved on SQuAD.

Model	**Dev**		**Test**	
	F1	Exact	F1	Exact
Match-LSTM[1]	48.9	35.2	48.0	33.4
BARB[2]	49.6	36.1	48.3	34.1
Neural BoW Baseline	37.6	25.8	36.6	24.1
FastQA $k = 5$	**56.4**	**43.7**	55.7	41.9
FastQAExt $k = 5$	56.1	**43.7**	**56.1**	**42.8**

Table 3: Results on the NewsQA dataset. [1]Wang and Jiang (2017) was re-implemented by [2]Trischler et al. (2017).

Results presented in Tables 2 and 3 clearly demonstrate the strength of the FastQA system. It is very competitive to previously established state-of-the-art results on the two datasets and even improves those for NewsQA. This is quite surprising when considering the simplicity of FastQA putting existing systems and the complexity of the datasets, especially SQuAD, into perspective. Our extended version FastQAExt achieves even slightly better results outperforming all reported results prior to submitting our model on the very competitive SQuAD benchmark.

In parallel to this work Chen et al. (2017) introduced a very similar model to FastQA, which relies on a few more hand-crafted features and a 3-layer encoder instead of a single layer in this

work. These changes result in slightly better performance which is in line with the observations in this work.

7.3 Do we need additional interaction?

In order to answer this question we compare FastQA, a system without a complex word-by-word interaction layer, to representative models that have an interaction layer, namely FastQAExt and the Dynamic Coattention Network (DCN, Xiong et al. (2017)). We measured both time- and space-complexity of FastQAExt and a reimplementation of the DCN in relation to FastQA and found that FastQA is about twice as fast as the other two systems and requires $2-4\times$ less memory compared to FastQAExt and DCN, respectively[6].

In addition, we looked for systematic advantages of FastQAExt over FastQA by comparing SQuAD examples from the development set that were answered correctly by FastQAExt and incorrectly by FastQA (589 FastQAExt wins) against FastQA wins (415). We studied the average question- and answer length as well as the question types for these two sets but could not find any systematic difference. The same observation was made when manually comparing the kind of reasoning that is needed to answer a certain question. This finding aligns with the marginal empirical improvements, especially for NewsQA, between the two systems indicating that FastQAExt seems to generalize slightly better but does not offer a particular, systematic advantage. Therefore, we argue that the additional complexity introduced by the interaction layer is not necessarily justified by the incremental performance improvements presented in §7.2, especially when memory or run-time constraints exist.

7.4 Qualitative Analysis

Besides our empirical evaluations this section provides a qualitative error inspection of predictions for the SQuAD development dataset. We analyse 55 errors made by the FastQA system in detail and highlight basic abilities that are missing to reach human level performance.

We found that most errors are based on a lack of either syntactic understanding or a fine-grained semantic distinction between lexemes with similar meanings. Other error types are mostly related to annotation preferences, e.g., answer is good but there is a better, more specific one, or ambiguities within the question or context.

Example FastQA errors. Predicted answers are underlined while correct answers are presented in boldface.

Ex. 1: *What religion did the Yuan discourage, to support Buddhism?*

Buddhism (especially Tibetan Buddhism) flourished, although **Taoism** endured ... persecutions... from the Yuan government

Ex. 2: *Kurt Debus was appointed what position for the Launch Operations Center?*

Launch Operations Center (LOC) ... Kurt Debus, a member of Dr. Wernher von Braun's ... team. Debus was named the LOC's first **Director** .

Ex. 3: *On what date was the record low temperature in Fresno?*

high temperature for Fresno ... set on July 8, 1905, while the official record low ... set on **January 6, 1913**

A prominent type of mistake is a lack of fine-grained understanding of certain answer types (Ex. 1). Another error is the lack of co-reference resolution and context sensitive binding of abbreviations (Ex. 2). We also find that the model sometimes struggles to capture basic syntactic structure, especially with respect to nested sentences where important separators like punctuation and conjunctions are being ignored (Ex. 3).

A manual examination of errors reveals that about 35 out of 55 mistakes (64%) can directly be attributed to the plain application of our heuristic. A similar analysis reveals that about 44 out of 50 (88%) analyzed positive cases are covered by our heuristic as well. We therefore believe that our model and, wrt. empirical results, other models as well mostly learn a simple context/type matching heuristic.

This finding is important because it reveals that an extractive QA system does not have to solve the complex reasoning types of Chen et al. (2016) that were used to classify SQuAD instances (Rajpurkar et al., 2016), in order to achieve current state-of-the-art results.

8 Related Work

The creation of large scale cloze datasets such the DailyMail/CNN dataset (Hermann et al., 2015)

[6] We implemented all models in TensorFlow (Abadi et al., 2015).

or the Children's Book Corpus (Hill et al., 2016) paved the way for the construction of end-to-end neural architectures for reading comprehension. A thorough analysis by Chen et al. (2016), however, revealed that the DailyMail/CNN was too easy and still quite noisy. New datasets were constructed to eliminate these problems including SQuAD (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2017) and MsMARCO (Nguyen et al., 2016).

Previous question answering datasets such as MCTest (Richardson et al., 2013) and TREC-QA (Dang et al., 2007) were too small to successfully train end-to-end neural architectures such as the models discussed in §4 and required different approaches. Traditional statistical QA systems (e.g., Ferrucci (2012)) relied on linguistic pre-processing pipelines and extensive exploitation of external resources, such as knowledge bases for feature-engineering. Other paradigms include template matching or passage retrieval (Andrenucci and Sneiders, 2005).

9 Conclusion

In this work, we introduced a simple, context/type matching heuristic for extractive question answering which serves as guideline for the development of two neural baseline system. Especially FastQA, our RNN-based system turns out to be an efficient neural baseline architecture for extractive question answering. It combines two simple ingredients necessary for building a currently competitive QA system: a) the awareness of question words while processing the context and b) a composition function that goes beyond simple bag-of-words modeling. We argue that this important finding puts results of previous, more complex architectures as well as the complexity of recent QA datasets into perspective. In the future we want to extend the FastQA model to address linguistically motivated error types of §7.4.

Acknowledgments

We thank Sebastian Riedel, Philippe Thomas, Leonhard Hennig and Omer Levy for comments on an early draft of this work as well as the anonymous reviewers for their insightful comments. This research was supported by the German Federal Ministry of Education and Research (BMBF) through the projects ALL SIDES (01IW14002), BBDC (01IS14013E), and Software Campus (01IS12050, sub-project GeNIE).

References

Martin Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Rajat Monga, Sherry Moore, Derek Murray, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Oriol Vinyals, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems .

Andrea Andrenucci and Eriks Sneiders. 2005. Automated question answering: Review of the main approaches. In *ICITA*.

Danqi Chen, Jason Bolton, and Christopher D Manning. 2016. A Thorough Examination of the CNN / Daily Mail Reading Comprehension Task. *ACL* .

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer Open-Domain Questions. *ACL* .

Hoa Trang Dang, Diane Kelly, and Jimmy J Lin. 2007. Overview of the TREC 2007 Question Answering Track. *TREC* .

D. A. Ferrucci. 2012. Introduction to "This is Watson". *IBM Journal of Research and Development* .

Yarin Gal and Zoubin Ghahramani. 2015. Dropout as a Bayesian Approximation : Representing Model Uncertainty in Deep Learning. *ICML* .

Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching Machines to Read and Comprehend. *NIPS* .

Felix Hill, Antoine Bordes, Sumit Chopra, and Jason Weston. 2016. The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations. *ICLR* .

Sepp Hochreiter and Jürgen Schmidhuber. 1997. LONG SHORT-TERM MEMORY. *Neural Computation* .

Diederik Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. *ICLR* .

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural Architectures for Named Entity Recognition. *NAACL* .

Peng Li, Wei Li, Zhengyan He, Xuguang Wang, Ying Cao, Jie Zhou, and Wei Xu. 2016. Dataset and Neural Recurrent Sequence Labeling Model for Open-Domain Factoid Question Answering. *arXiv:1607.06275v1 [cs.CL]* .

Dan I. Moldovan, Sanda M. Harabagiu, Marius Pasca, Rada Mihalcea, Richard Goodrum, Roxana Girju, and Vasile Rus. 1999. Lasso: A tool for surfing the answer net. In *TREC*.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. Ms Marco: a Human Generated Machine Reading Comprehension Dataset. *NIPS* .

Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. GloVe: Global Vectors for Word Representation. *EMNLP* .

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ Questions for Machine Comprehension of Text. *EMNLP* .

Matthew Richardson, Christopher J C Burges, and Erin Renshaw. 2013. MCTest: A Challenge Dataset for the Open-Domain Machine Comprehension of Text. *EMNLP* .

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. 2017. Bi-Directional Attention Flow for Machine Comprehension. In *ICLR*.

Yelong Shen, Po-Sen Huang, Jianfeng Gao, and Weizhu Chen. 2016. ReasoNet: Learning to Stop Reading in Machine Comprehension. *arXiv:1609.05284* .

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. 2017. NewsQA: A Machine Comprehension Dataset. *arXiv:1611.09830* .

Shuohang Wang and Jing Jiang. 2017. Machine Comprehension Using Match-LSTM and Answer Pointer. In *ICLR*.

Zhiguo Wang, Haitao Mi, Wael Hamza, and Radu Florian. 2017. Multi-Perspective Context Matching for Machine Comprehension. *arXiv:1612.04211* .

Caiming Xiong, Victor Zhong, and Richard Socher. 2017. Dynamic Coattention Networks for Question Answering. *ICLR* .

Zhilin Yang, Bhuwan Dhingra, Ye Yuan, Junjie Hu, William W Cohen, and Ruslan Salakhutdinov. 2017. Words or Characters? Fine-grained Gating for Reading Comprehension. *ICLR* .

Yang Yu, Wei Zhang, Kazi Hasan, Mo Yu, Bing Xiang, and Bowen Zhou. 2017. End-to-End Reading Comprehension with Dynamic Answer Chunk Ranking. In *ArXiv*.