Diverse Ensembling for Question Answering

Benjamin Cohen-Wang, Edward Lee
Department of Computer Science
Stanford University
{bencw, edlee1}@stanford.edu

Abstract

In this paper we explore ensembling of different question answering systems, which significantly improves performance over any individual model. We propose a diverse ensemble of variants of three high-performing SQuAD model "families": the BiDAF Network [4], the Mnemonic Reader [2], and ReasoNet [5]. Our results support the claim that diverse ensembles of models, such as an ensemble of one model from each family, generally outperform less diverse ensemble of high-performing models, such as an ensemble of three models from the same high-performing family. Our final ensemble of three models from each family gets an F1 score of 79.5 and an EM score of 70.0 on the validation set.

1 Introduction

Machine Comprehension (MC) and Question Answering (QA) tasks have grown in popularity in recent years due to many new developments in Natural Language Processing (NLP) and the creation of large question answer datasets. Many high-performing MC models have been created that perform on near-human capability in tests using datasets like the Stanford Question Answering Dataset (SQuAD) [3]. One of the main threads throughout model performance has been the idea of an ensemble. Namely, by training multiple models with different initializations, performance can improve by anywhere from 1-3%, as different initializations allow models to learn slightly different representations and thus slightly different solutions for each datapoint [1, 4]. In this paper, we attempt to take it a step further, ensembling not just models with different initializations, but also models with different hyperparameters and different mechanisms entirely.

2 Related Work

The model we propose is largely based on variants of existing high-performing SQuAD models. In particular our model heavily relies on the bidirectional attention flow layer discussed in *Bi-Directional Attention Flow for Machine Comprehension* which produces a query-aware context layer that incorporates both context-to-question and question-to-context attention [4]. We also emulated and implemented variants of the iterative reasoning techniques described in *Reinforced Mnemonic Reader for Machine Comprehension* [2] and *ReasoNet: Learning to Stop Reading in Machine Comprehension* [5]. Finally, we drew ideas for our ensemble from *Neural Network Ensembles* [1], which discusses the benefits of constructing an ensemble of diverse models.

3 Model

Our model consists of an ensemble of nine different models, broken down into three model families with individual models within each family trained with different hyperparameters. By training different types of models with different sets of hyperparameters we increase the diversity of our ensemble, which has been shown to improve performance [1].

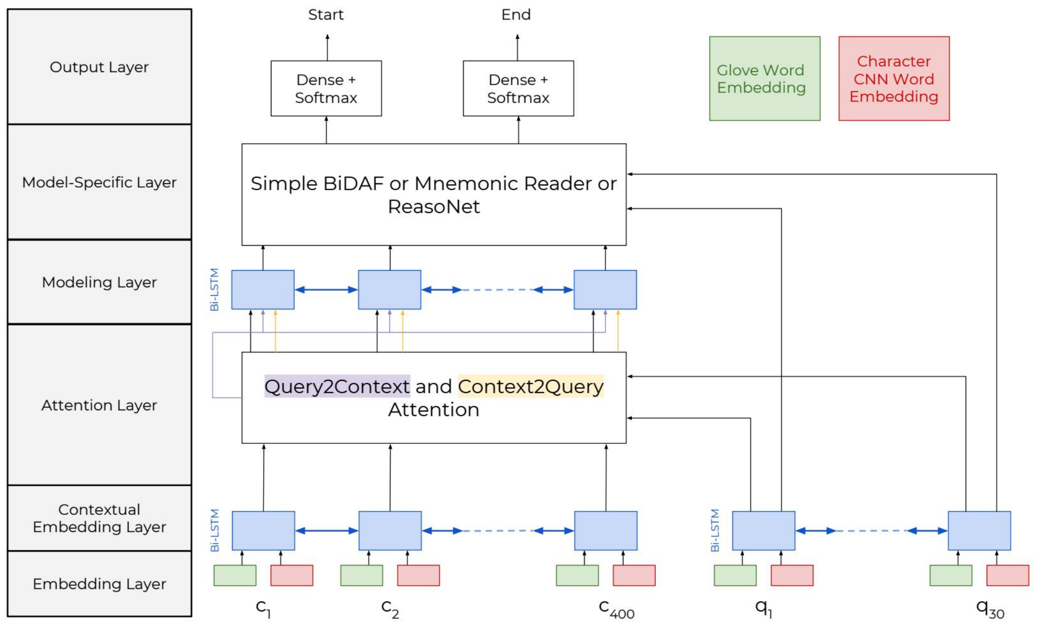

Figure 1: BiDAF Base Model

The three models use the same architecture to encode a query-aware context representation, but each run different operations on the output of this shared architecture to produce final distributions for the start and end positions of the answer.

3.1 Common BiDAF Layer

The three model families share the embedding, contextual embedding, attention, and modeling layers defined by the BiDAF network [4]. We chose to incorporate these layers as the base architecture for our model families because it effectively models interactions between the context and query with only one attention layer. Though two of our model families further relate the context to the query, we found that passing a query-aware context initially improves performance even for these models. The following visualizes this common architecture and its relationship to the model-specific layer that follows.

Embedding Layer: The embedding layer represents each word in the context and query as a high-dimensional vector by concatenating its pre-trained 100-dimensional GloVe embedding with 100-dimensional character-level word embeddings. These character-level embeddings are computed by applying two convolution layers to 20-dimensional character embeddings and max-pooling over the characters of every word.

Contextual Embedding Layer: The contextual embedding layer uses a bidirectional-LSTM with 200-dimensional hidden states, using individual word embeddings as inputs and sharing parameters between the context and query. The resulting outputs incorporate interactions between different words.

Attention Layer: The attention layer generates a query-aware representation of the context is the length of the context) by computing both context-to-query and query-to-context attention and blending these with the context outputs of the contextual embedding layer. The vanilla version of bi-directional attention flow is used, both to compute attention and for blending to produce output $G \in \mathbb{R}^{8d \times L_c}$, where $d = 200$ is the size of the bidirectional-LSTM hidden states and $L_c = 400$ is the length of the context.

Modeling Layer: Similarly to the contextual embedding layer, the modeling layer incorporates interactions between different words through a bidirectional-LSTM with 200-dimensional hidden states. However, the modeling layer acts only upon the query-aware context representation, rather than each of the context and query independently.

The output of the modeling layer $M \in \mathbb{R}^{2d \times L_c}$, and with each columns representing one word, is fed into model-specific layers discussed in the next section.

3.2 Three Model Families

3.2.1 Simple BiDAF

The first model family is based on the BiDAF network presented by *Bi-Directional Attention Flow for Machine Comprehension* [4], which includes a second modeling output M^2 computed by passing the common modeling layer M through another bidirectional-LSTM. The distributions of the start and end positions are computed as

$$p_{\text{start}} = \text{softmax}(w_{\text{start}}^T [G, M_{\text{start}}])$$
$$p_{\text{end}} = \text{softmax}(w_{\text{end}}^T [G, M_{\text{end}}])$$

where $w_{\text{start}} \in \mathbb{R}^{10d}$ and $w_{\text{end}} \in \mathbb{R}^{10d}$ are trainable weight vectors. The three variants of this model family involved training models with $M_{\text{start}} = M$ and $M_{\text{end}} = M^2$ (as described in the paper), with $M_{\text{start}} = M_{\text{end}} = M^2$, and with $M_{\text{start}} = M_{\text{end}} = M$ (without computing M^2).

3.2.2 Mnemonic Reader

The second model family is based on network used in *Reinforced Mnemonic Reader for Machine Comprehension* [2], an iterative reasoning model. We included both the iterative aligner and the memory-based answer pointer in our variant, which are the core iterative components of the model.

Iterative Aligner: The iterative aligner takes the output of the common modeling layer and applies an interactive alignment step, self alignment step, and an aggregation step over T iterations. At iteration t, the interactive aligner computes a coattention matrix B^t defined as $B^t_{ij} = q_i^T c_j$ where q_i and c_j are the current representations of the i'th query word and j'th context word, respectively. The attention output for c_j is then computed as

$$[q_1, q_2, ..., q_{L_q}] \cdot \text{softmax}(B^t_j)$$

and incorporated back into the context word representations through a Semantic Fusion Unit (SFU), introduced by the paper. The self alignment step is structured similarly to the interactive step but with a coattention matrix defined as $B^t_{ij} = c_i^T c_j$ and an attention output

$$[c_1, c_2, ..., c_{L_c}] \cdot \text{softmax}(B^t_j)$$

to capture interactions between context words rather than context and query words. Finally, the aggregation step passes the context representation through a bidirectional-LSTM.

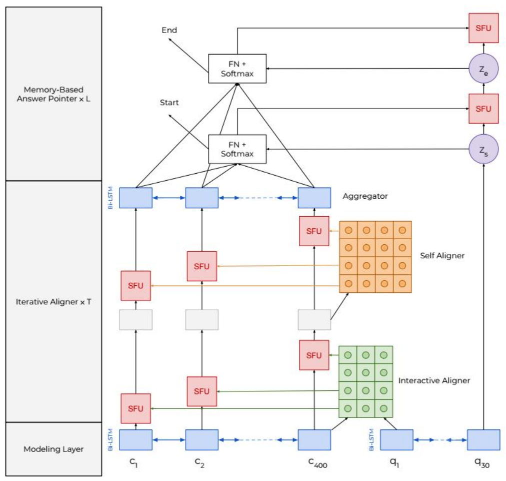

Figure 2: Mnemonic Reader-Specific Layer

Over multiple iterations, the iterative aligner allows the model to capture interactions between the context and the query, interactions between different and potentially distant context words, and temporal interactions between query and self aware context words. We used two iterations of the aligner. We differentiated our mnemonic reader from that described in the paper by preceding interactive alignment with the common BiDAF layer, which allows the first iteration of the aligner to operate on a query-aware context and improved performance over our original implementation.

Memory-Based Answer Pointer: The iterative aligner is followed by the memory-based answer pointer, which iteratively determines the start and end position probability distributions by alternating between the two. It conditions the start and end distributions on each other by maintaining memory initialized to the final query-state. The start distribution of the l'the step of L total steps with memory z_s^l is determined as

$$s^l = FN([c_1, ..., c_{L_c}], z_s^l, [c_1, ..., c_{L_c}] \circ z_s^l)$$
$$p_{\text{start}}^l = \text{softmax}(w_s^l s^l)$$

where FN represents some feedforward neural network, and w_s^l trainable weights. An evidence vector $u_s^l = [c_1, ..., c_{L_c}] \cdot p_{\text{start}}^l$ is than used to fed into an SFU along with z_s^l to produce the next memory state z_e^l. z_e^l is then used to determine p_{end}^l analogously to z_s^l and p_{start}^l, and if $l < L$ is also used to determine the start memory state for the next iteration z_s^{l+1}. The start and end distributions of the final iteration are those outputted by the model.

The three variants of the mnemonic reader model family were models with zero, one, and two hidden layers within FN used to determine the start and end distributions.

3.2.3 ReasoNet

The third and final model family is based on *ReasoNet: Learning to Stop Reading in Machine Comprehension* [5], another iterative reasoning model.

Using the common modeling layer M as input, ReasoNet makes predictions by storing an internal state s_t initialized to the final query-state like mnemonic reader and on each iteration t applying a termination gate to determine whether to continue iteration. If so, the an attention output x_t is computed based on the interaction between s_t and M, which is then used to determine the next state s_{t+1}. Otherwise, or if the maximum number of steps T_{max} is reached, the start and end distributions are predicted by answer module. Specific implementation details are as follows.

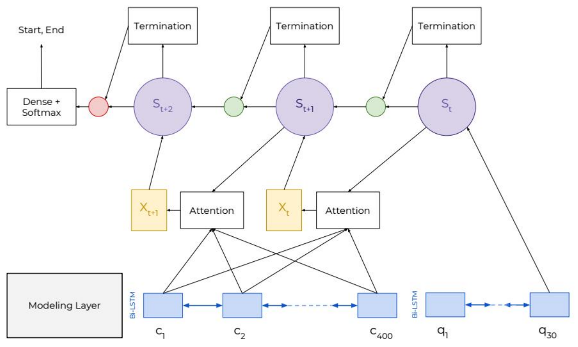

Figure 3: ReasoNet-Specific Layer

Termination Module: The termination decision is sampled from $f_{\text{tg}}(s_t) = \sigma(W_{\text{tg}} s_t + b_{\text{tg}})$ where W_{tg} and b_{tg} are trainable parameters.

Attention Module: The attention vector x_t between s_t and M, is computed as $x_t = M \cdot \text{softmax}(\gamma[\cos(W_1 M_i, W_2 s_t), i \in \{1, ..., L_c\}])$, where W_1 and W_2 are trainable and $\gamma = 10$ as presented by the paper.

Internal State Controller: A bidirectional-LSTM with s_t as its hidden state and x_t as its input updates the internal state to s_{t+1}.

Answer Module: The final answer is determined by applying a densely connected layer and softmax layer for each of start and end to $[M, M \circ s_t]$, a blended representation of the internal state and memory found to work well. The softmax layer produces the desired distributions p_{start} and p_{end}.

Our first implementation of the ReasoNet model is the vanilla version presented in the paper. Our second implementation was a variant which modified the answer module to pass the blended representation through a bidirectional-LSTM before applying the dense and softmax layers. The final variant included both a bidirectional-LSTM pass and a fully connected layer bfore the dense and softmax layers. The purpose of these additional layers was to further incorporate interactions between the final internal state and different parts of the context, and improved performance over our implementation of the vanilla version.

3.3 Ensemble

Each of the nine models generates probability distributions for the start and end positions of the answer. We ensemble these predictions to produce the final distributions by computing their average. More specifically, letting $p_{\text{start}_i} \in \mathbb{R}^C$ and $p_{\text{end}_i} \in \mathbb{R}^C$ be the start and end distributions of model i, for $i = 1, ..., N$ where $N = 9$ is the number of models, the ensembled probability distributions are

$$p_{\text{start}} = \sum_{i=1}^{N} w_{\text{start}_i} p_{\text{start}_i} \qquad p_{\text{end}} = \sum_{i=1}^{N} w_{\text{end}_i} p_{\text{end}_i}$$

where the sum and weight multiplication are computed element-wise, and where $w_{\text{start}} \in \mathbb{R}^N$ and $w_{\text{end}} \in \mathbb{R}^N$ are weights. In practice, we have found equal weighting for all nine models with $w_{\text{start}_i} = w_{\text{end}_i} = 1/N$, likely because our nine single models perform comparably. However, we have not formally optimized these weights, which we believe could yield even better results.

3.4 Smart Span Selection

Even though our two iterative reasoning model families condition the probability distributions for the start and end positions on each other, since the ensemble averages distributions from different models relationships between these two distributions are not necessarily maintained. As such we decided to condition the end distribution of the final ensemble on the start distribution by selecting start index s and end index e which maximize the product $p_{\text{start}_s} p_{\text{end}_e}$ subject to the constraint that s and e belong to the same sentence. We chose to apply this constraint because the large majority of answers are contained by a single sentence and this constraint prevents long incorrect answers resulting from choosing s and e independently.

Applying smart span selection to our final ensemble model resulted in an F_1 increase of 1.26% and an EM increase of 0.95% on the validation set.

4 Experiments

4.1 Dataset

SQuAD[3] is a question-answer dataset over Wikipedia articles with over 100,000 questions. Of particular note is that the answer is in substring of the context. Multiple human answers are given, and the score is added if the model matches any. Two metrics are used to evaluate models on this dataset: Exact Match (EM) and a more relaxed metric F1, which measures precision and recall.

4.2 Model Details

The nine models were trained with Tensorflow's Adam Optimizer with learning rate 0.001, batch size of 32, between 20,000 and 30,000 epochs. A maximum question length of 30 words, a maximum context length of 400 words, and a maximum word length of 20 characters were used, with larger inputs being truncated. A 200-dimensional hidden state was used for all LSTMs. For regularization, dropout was applied to each LSTM, to any fully connected layers, and between CNN layers for the character-level word embeddings. For our implementation of the Mnemonic Reader and ReasoNet, we used the iteration parameters cited: $T = 2$ for iterative alignment, $L = 2$ for the memory-based answer pointer, and $T_{\text{max}} = 10$ for the ReasoNet state controller.

5 Results and Analysis

Our best single model ReasoNet scores an F1 of 76.4 and an EM of 66.7 on the validation set, while our full ensemble of 3 models from each model family achieves an F1 of 79.5 and an EM of 70.0. Our results relative to other high-performing models can be seen in Table 1.

[1] Due to issues with CodaLab submissions, these numbers were generated on the validation set.

Table 1: Model Performance on validation/test set vs references

	F1	**EM**
CS224N Baseline	44.2	35.1
Logistic Regression Baseline	51.0	40.4
Our ReasoNet (Single)[1]	**76.4**	**66.7**
BiDAF (Single)	77.5	68.4
ReasoNet (Single)	79.4	70.6
Our Full Ensemble[1]	**79.5**	**70.0**
BiDAF (Ensemble)	81.1	73.3
Reinforced Mnemonic (Single)	81.8	73.2
ReasoNet (Ensemble)	82.5	75.0
Reinforced Mnemonic (Ensemble)	84.9	77.7

Table 2: Ensemble Ablations on validation set (all our own implementations)

	F1	**EM**
Mnemonic Reader (Single)	75.9	65.1
BiDAF (Single)	76.0	65.5
ReasoNet (Single)	76.4	66.7
Mnemonic Reader (Ensemble)	77.9	68.0
3x BiDAF (Ensemble)	78.2	68.4
3x ReasoNet (Ensemble)	78.8	69.4
1x BiDAF/Mnemonic/ReasoNet (Ensemble)	78.8	69.5
3x BiDAF/Mnemonic/ReasoNet (Ensemble)	**79.5**	**70.0**

5.1 Ablations

Listed in Table 2 is the performance of different combinations of single models and ensembles. The ensemble performances of a single model family reflect their single model relatives, with F1 increasing by around 2% for each family. Additionally, the 1x B/M/R ensemble that takes in only one model from each model family is able to perform at or above the level of the best-performing single model family ensemble, namely 3x ReasoNets, which consists of a combination of 3 ReasoNets.

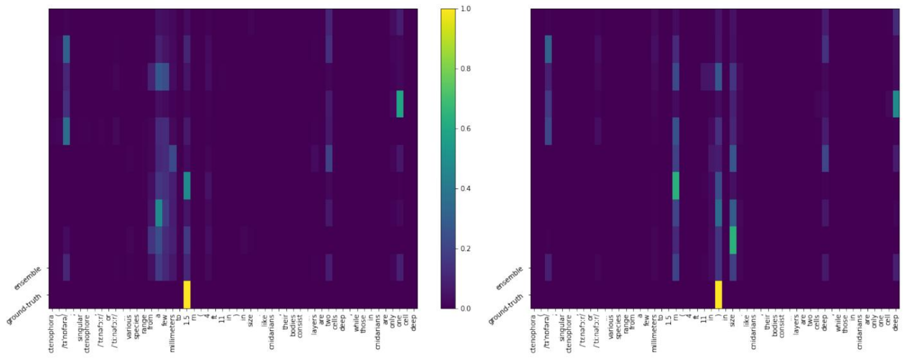

Figure 4: Select start (left) and end (right) probability distributions for a sample where ensemble is able to correctly predict the answer, while each individual model is not. Second row from the bottom is the ensemble, and the bottom-most is the ground-truth.

5.2 Ensemble Sample

To analyze how the ensemble chooses its final answer, we visualize the probability distributions of each model for a given (context, question) pair in Figure 4. This particular sample was chosen since each single model was unable to answer the question exactly, but combined, the ensemble was able to choose the correct start and end span. From the figure, we can see that several of the probability distributions have selected incorrect answer spans with relatively high confidence. Of particular note is the 4th single model, which appears to locked in on an incorrect answer with a confidence of above 0.5 for both start and end. However, by averaging all the distributions, we are able to ignore this outlier (and many others), and choose the correct answer.

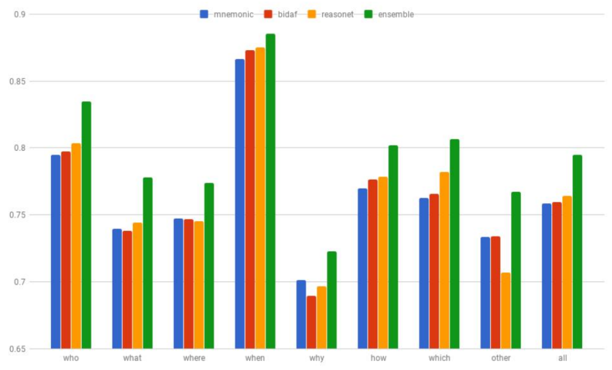

Figure 5: F1 performance of best single models and best ensemble over different question types. (Question types are selected based on which word appears first in the sentence.)

5.3 Question Types

The model's F1 performance, as well as the F1 performance of the best single model of each family, on each question type is visualized in Figure 5. We see that models tend to have the same relative performance on each question type, performing best on ”When” and worst on ”Why”. This is likely because answers to ”When” are have very clear boundaries and almost definitely are related to time (e.g. ”Sunday”, ”week”, etc.). However, ”Why” questions likely have no such guarantees, as understanding when to stop can be difficult, and answer lengths for ”Why” are likely much longer than answer lengths for ”When”. We also see that ensembling seems to give diminishing returns as single models get better, unable to help much with the ”When” questions, but improving other question types like Other and ”Who” significantly.

5.4 Error Analysis

We analyze error by examining specific samples where the model fails to choose the exact answer on the validation set. We find two main errors with the model: syntactic complications and imprecise boundaries.

5.4.1 Imprecise Boundaries

Most of the errors seemed to come from imprecise boundaries, namely the model would include or exclude several words from either end. Either improving the model or adding more features to our span selection may help with this.

- **Context:** In 1542, Luther read a Latin translation of the Qur'an. ... He opposed banning the publication of the Qur'an, wanting it exposed to scrutiny.
- **Question:** What purpose would Luther have in not wanting to ban the Qur'an?
- **Answers:** [‘exposed to scrutiny.’, ‘wanting it exposed to scrutiny.’, ‘exposed to scrutiny’]
- **Prediction:** ’scrutiny’

5.4.2 Syntactic Complication

Another major errors was that of syntactic complications. Namely, in much of the time, similar words in a span of the context and the question increase the likelihood of the answer lying within that span. However, there may be other sentences that have similar words that do not actually include the answer.

- **Context:** On the next play, Miller stripped the ball away from Newton, and after several players dove for it, ... Then Anderson scored on a 2-yard touchdown run and Manning completed a pass to Bennie Fowler for a 2-point conversion ...
- **Question:** What player punched the ball in from the 2?
- **Answers:** [‘Anderson’, ‘Anderson’, ‘Anderson’]
- **Prediction:** ’Miller’

In this case, ‘punched the ball in from’ is very similar to ‘stripped the ball away from’, which is likely why ‘Miller’ gets picked, and not ‘Anderson’. It is likely pretty difficult to address these errors, as they run against much of the training set and could be considered adversarial examples.

6 Conclusion

In this paper, we implement three variants on three different types of question-answer models: Bi-Directional Attention Flow, Mnemonic Reader, and ReasoNet. We then ensemble all 9 different models to achieve competitive results in the Stanford Question Answering Dataset (SQuAD). Ablation tests on the ensemble suggest that combining more diverse models that may not be as high-performing can perform at or above high-performing but similar models. Further work can involve experimenting with different techniques to ensemble like max vote or a neural network, or attempting to make the model resistant to adversarial attacks.

Acknowledgements

We would like to thank Richard Socher and the CS224N TAs for their helpful comments on improving models, as well as a fascinating default project topic. We would also like to thank Microsoft for providing Azure resources.

References

[1] L. K. Hansen and P. Salamon. “Neural Network Ensembles”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 12.10 (Oct. 1990), pp. 993–1001. ISSN: 0162-8828. DOI: 10.1109/34.58871. URL: http://dx.doi.org/10.1109/34.58871.

[2] Minghao Hu, Yuxing Peng, and Xipeng Qiu. “Mnemonic Reader for Machine Comprehension”. In: *CoRR* abs/1705.02798 (2017). arXiv: 1705.02798. URL: http://arxiv.org/abs/1705.02798.

[3] Pranav Rajpurkar et al. “SQuAD: 100, 000+ Questions for Machine Comprehension of Text”. In: *CoRR* abs/1606.05250 (2016). arXiv: 1606.05250. URL: http://arxiv.org/abs/1606.05250.

[4] Min Joon Seo et al. “Bidirectional Attention Flow for Machine Comprehension”. In: *CoRR* abs/1611.01603 (2016). arXiv: 1611.01603. URL: http://arxiv.org/abs/1611.01603.

[5] Yelong Shen et al. “ReasoNet: Learning to Stop Reading in Machine Comprehension”. In: *CoRR* abs/1609.05284 (2016). arXiv: 1609.05284. URL: http://arxiv.org/abs/1609.05284.