Conditional Iterative Refinement for Question Answering

Yu-Chung Hsiao
yuchsiao@stanford.edu

Abstract

For the SQuAD task, the common start and end probability prediction strategy may not be the thorough use of information. Additional subtasks and labels can be readily derived from the given answer spans. We propose a novel conditional iterative refinement method that can directly consume all subtask information and outperform traditional multitask learning with hard parameter sharing. A corresponding training scheme is developed for improving training convergence. We observed a 2.0% improvement for the average F-1 score and a 2.5% improvement for the exact match metric, achieving 80.65 and 77.76, respectively.

1 Introduction

The SQuAD dataset [1], since its debut, has sparked a lot of interests from the community and stimulated countless research ideas. Most of the works, despite different designs of deep learning architectures, are supervised learning by minimizing the sum of two cross-entropy loss functions: the start probability and the end probability of the answer span (e.g., [2, 3, 4, 5, 6, 7]). The final predicted answer span is then determined by maximizing the product of the two probabilities, conditioned on some hand-crafted criteria, such as maximum answer length. The SQuAD 2.0 dataset [8], in addition to the single span prediction, removes the always-answerable assumption and requires the model to be able to detect such unanswerable cases. To build on top of the start and end prediction, one common approach is to add a dummy token to the sequence: when the predicted span is exactly the dummy token, the question is determined not answerable [9, 10].

These approaches are effective in confining the answer to a single text span and, at the same time, casting two objectives (span prediction and has-an-answer prediction) into a unified solution. However, transforming answer spans exclusively into start and end positions may not be the thorough use of information. Other types of labels, such as whether a word is part of the answer, whether a name entity is the answer, and whether the context-question pair is answerable, are readily available to form related classification problems, either at the token level or the whole sequence level. In other words, there are many more ways to expand a single answer span into multiple sets of labels, each of which can be used as an auxiliary subtask to assist the main, start and end position prediction task. This naturally leads to a multitask learning problem.

Intuitively, adding more related labels enriches the information that a model can learn. However, a recent report [11] shows that while some tasks benefit from multitask learning, some may be severely interfered and perform much more poorly with the presence of others. This negative effect is especially apparent when the output layer is responsible for very different types of predictions, such as language generation for machine translation versus pointer networks for SQuAD. Since it is difficult to assess whether an extra subtask is constructive beforehand, adding auxiliary subtasks to the main task is usually not a preferable choice.

To address this issue, we propose an iterative refinement scheme on top of the multitask setting. If a subtask reveals a new and related perspective of the main task, then consuming the knowledge of subtask labels should be directly constructive to the main task. One explanation why this is not always observed in multitasking learning via hard parameter sharing (a shared model with only the

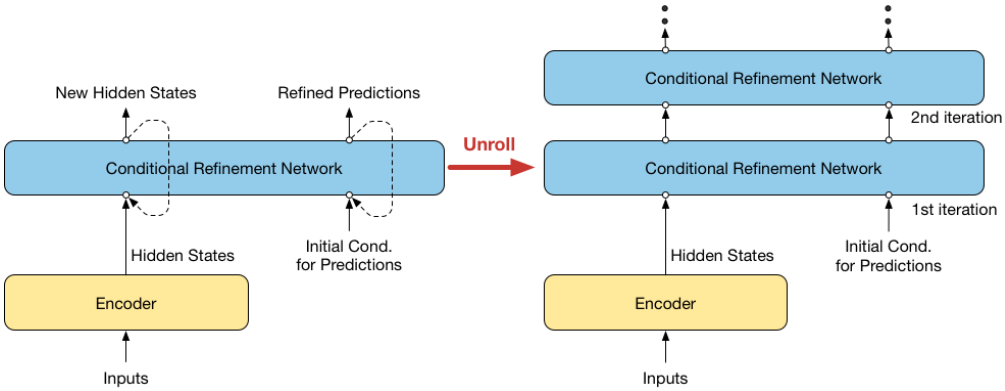


Figure 1: Concept of conditional iterative refinement.

output part specialized for each subtask) is that knowledge of subtask labels is *not* directly consumed but more of implicitly "inferred" by backpropagation. In contrast, we propose to use a recurrent layer which is explicitly conditioned on the previous predictions of all subtasks to generate the next round of predictions. This conditional iterative refinement process is depicted in Figure 1.

Specifically, we define two tasks as the auxiliary subtasks to the main start and end prediction problem: 1) token-level binary classification problem and 2) sequence-level has-an-answer binary classification problem. With the synergy of these subtasks and conditional iterative refinement, the new model achieves a 2.5% lift for exact match and a 2.0% lift for average F-1 score, reaching absolute scores 77.76 and 80.65, respectively. Without conditional iterative refinement, we observed performance drops by including these auxiliary subtasks. This coincides our hypothesis above and we will discuss the experiment details in Section 4.

This paper organizes as follows: Section 2 discusses related works, Section 3 covers our architectural design, along with the training and inference strategies, Section 4 details experiments. We will provides some qualitative analyses in Section 5, concluding remarks and future works in Section 6 and Section 7, respectively.

2 Related Works

2.1 Iterative Refinement

Several works on the topic of iterative refinement in deep learning have been done for natural language processing, such as [12], [13], and [14]. These work all focused on natural language generation as opposed to sequence classification and tagging problems as SQuAD. One key distinction is that in natural language generation, the decoding step is commonly trained by one-step supervision, without involving recursion or iterative processes. In this work, we include such iterative processes as part of training.

2.2 Denoising Autoencoder

After the first iteration of the conditional refinement network (Figure 1), the hidden states and the predicted outputs can be considered the noisy version of those from the next iteration. The expectation is that each iteration can denoise the outputs from the previous iteration. From this perspective, this work is related to denoising autoencoder [15] except for its supervised training. Although it has not been explored yet, the theoretical foundation established in [16] can be potentially useful for future works.

2.3 Multitask Learning

As we define multiple subtasks and encourage a single model to learn across various labels via separate output layers, this approach is related to multitask learning with hard parameter sharing [17]. The iterative refinement process is conceptually similar to the recurrent neural network. However, since we only have one set of reference labels but need to make sure the model works for multiple iterations,

we progressively increase the number of iterations involved in training. This progressive training strategy is conceptually related to the widening training procedure introduced in [18], although the widening step in [18] is applied to splitting networks progressively.

3 Approach

3.1 Subtasks

We define two additional subtasks to complement the main start and end position prediction task. Note that the labels of these two subtasks can be directly constructed from the answer spans of the original SQuAD training data without further human labor annotations.

3.1.1 Token-level binary classification (TL)

Given a pair of a context paragraph and a question, if a token is included in the answer span, then labeled as 1, otherwise, 0. If the context-question pair has no answer, then the dummy token defined in in [9] is labeled as 1 to be consistent with the start and the end position main task.

The idea behind this subtask is to encourage the model to learn all the tokens between the start and the end positions of the answer span in the hope that even when the model fails to predict the most correct start and end positions, it will still be able to predict the words in between as alternative boundaries. Intuitively, this will help the F-1 metric but may not be very useful for the exact match metric.

3.1.2 Sequence-level has-an-answer binary classification (HA)

Given a pair of a context paragraph and a question, if there is no span in the context that answers the question, then this context-question pair is labeled as 0, otherwise, 1.

This is to make the official has-an-answer SQuAD 2.0 task a standalone subtask, as opposed to being embedded in the start and end position prediction. We found that having this standalone subtask is really useful not only for training but for inference, especially when the percentage of no-answer cases differs from the training data.

3.2 Architecture

We propose a deep learning architecture in Figure 2 to implement the concept of the conditional iterative refinement.

3.2.1 Encoder

We use the pre-trained BERT [9], including its WordPiece tokenizer, as the encoder. We also follow the BERT's approach to include the question followed by the context paragraph, with a separator in between. The whole sequence is prepended a `CLS` token for accommodating the no-answer case.

3.2.2 Matching and adaptor layer

The matching and adaptor network is to provide a transition from the typically large hidden size of a pre-trained encoder to a relatively small conditional layer. This layer is non-iterative and can be seen as an extension of the encoder. For size adaptation, a simple linear layer suffices. If extra matching capacity is needed, a bi-LSTM layer can be used. Note that if neither adaptation nor extra matching is necessary, this layer is entirely optional.

3.2.3 Conditional refinement network

The conditional refinement network is the iterative unit with shared weights across iterations. It comes with two inputs: hidden states on the left and conditions on the right, and two outputs: hidden states on the left and output logits (values before softmax) on the right. The output hidden states and the output logits are then fed back to the network itself as the inputs of the next iteration.

Mathematically, the conditional refinement network attempts to model, for iteration r,

$$p(H_r, C_r \mid H_{r-1}, C_{r-1}), \tag{1}$$

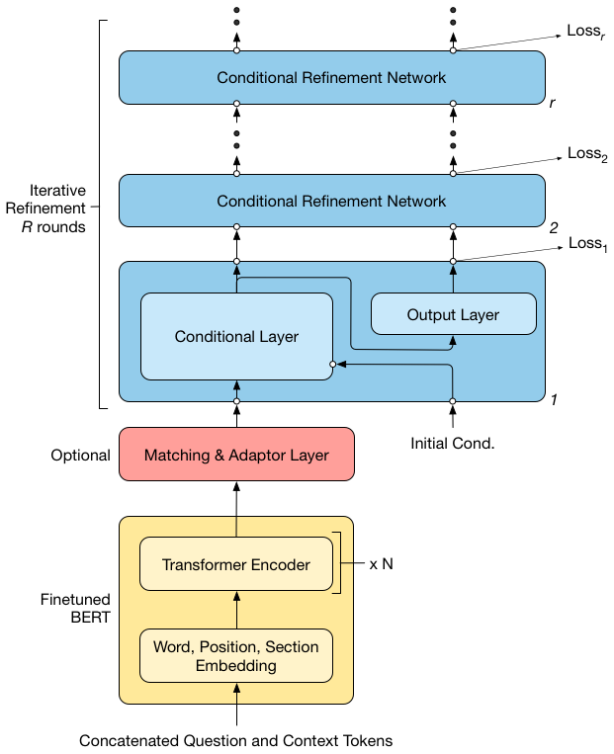


Figure 2: Design of the conditional iterative refinement network.

given H_0 from the encoder and the learnable initial conditions C_0, where H represents hidden states and C conditions for output logits, such that the summation of loss $\mathcal{L}$ of $y_i^r \sim C_r$ against empirical samples $\tilde{y}_i$

$$\sum_i \mathcal{L}(y_i^r, \tilde{y}_i) \tag{2}$$

is minimized. To simplify (1), we further assume the following conditional independence

$$(C_r \perp\!\!\!\perp H_{r-1}) \mid H_r \quad \text{and} \quad (C_r \perp\!\!\!\perp C_{r-1}) \mid H_r. \tag{3}$$

Then (1) can be factored as a product of two factors

$$p(C_r \mid H_r)\, p(H_r \mid H_{r-1}, C_{r-1}) \tag{4}$$

We will use the conditional layer to model the second factor and the output layer to model the first factor, as described below.

3.2.4 Conditional layer

A simple design of the conditional layer can be a multi-layer Transformer encoder [19] with that modification that the input takes the concatenation of hidden states and output logits. It is straightforward when the output logits are from a token-level subtask. The concatenation is then token by token. For a sequence-level subtask, from which there is only one logit for the whole sequence tokens, we repeat the identical logit for every token of the sequence.

3.2.5 Output layer

The output layer consists of multiple outputs, each of which is dedicated to a particular subtask. For each subtask, we choose to use two linear layers with a ReLU activation function in between.

3.3 Training

The overall loss function is the summation of the losses for the main task and all auxiliary subtasks with a weight λ

$$\mathcal{L} = \mathcal{L}_{\text{Start}} + \mathcal{L}_{\text{End}} + \lambda(\mathcal{L}_{\text{TL}} + \mathcal{L}_{\text{HA}}), \tag{5}$$

each of which represents tasks for start position, end position, token-level classification, and has-an-answer classification. The weight λ is introduced to balance the importance between the main task and the subtasks

During training, we start with a single refinement iteration, i.e., running (4) only once $r = 1$. Once the overall model starts to learn, we then progressively increase the number of refinement iterations. This is because, unlike typical decoder training processes, where each step has its own supervision, the conditional iteration refinement utilizes the loss only at the last iteration. In other words, relying on the backpropagation to pass the supervision information through several iterations of random initialized conditional layer is practically not convergent. Besides, when the number of refinement iterations $r > 1$ increases, we require, with a fixed probability, the model is trained with fewer refinement iterations $r' < r$ that it was trained before. This helps the model properly functions for a wide range of numbers of refinement iterations. We have observed, without randomly trained at fewer refinement iterations, the model may forget and collapse for lower number iterations that it has once learned. When this forgetting condition happens, the model no longer benefits from utilizing knowledge across subtasks. We will return to this point in Section 4.

3.4 Inference

The multiple subtask outputs provide extra flexibility of how the final answer spans should be extracted. For instance, one can "refine" the span by trimming out words at the start or the end of the span that have low probability from the token-level classification. Even more, one can completely discard the answer span if the harmonic mean of the token-level probabilities is below a certain threshold. The iterative refinement further certifies the stability of the span prediction over iterations. It is also possible to use the prediction of the has-an-answer subtask (HA) to jointly determine whether the context-question pair really has an answer. In fact, as we will see in Section 4, setting a threshold for the sigmoid of HA logit is a more tunable control knob than the dummy token from the start and the end position prediction.

4 Experiments

4.1 Data

Default project dataset: SQuAD v2.0. No other dataset or data augmentation is used.

4.2 Evaluation Method

We use the standard Exact match (EM) and F-1 score (F1) defined in SQuAD v2.0 [8]. In addition, we use "AvNA" to denote the accuracy for Answer vs. No Answer, "TL" for the ROC-AUC for the TL subtask and "HA" for the ROC-AUC for the HA subtask as in Table 1.

4.3 Baseline Model

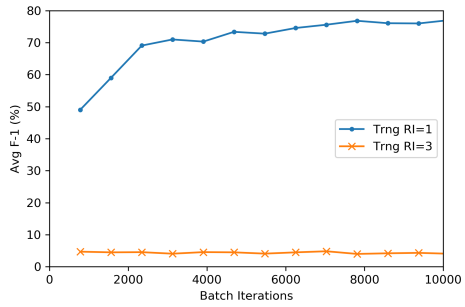
The baseline is the basic BERT model trained from the training script (`run_squad.py`) from the Huggingface implementation[1] for the SQuAD v2.0 dataset. All hyperparameters are unaltered. The performance is shown in Table 1 in the row of "HF BERT."

To show the effectiveness of the proposed Conditional Iterative Refinement (CIR), we also use the non-iterative version of CIR, i.e., $r = 1$, which is simply the traditional MultiTask Learning (MTL) with hard parameter sharing. The performances of MTL, with and without TL and HA, are summarized in the second row set of Table 1.

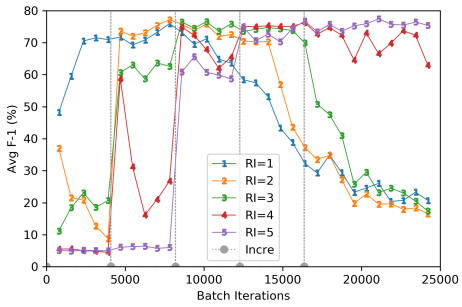
4.4 Model Parameters

We use both BERT base uncased and large uncased pre-trained models in our experiments. As in Table 1, the large-size model is denoted by "L" at the beginning of the experiment name. Otherwise, the base-size is used. Regardless of the BERT model size, maximum sequence length is set to 448, and the maximum query length is set to 64. The top, the 4th from the top, and the 9th from the top hidden states of BERT are concatenated as the final output of BERT.

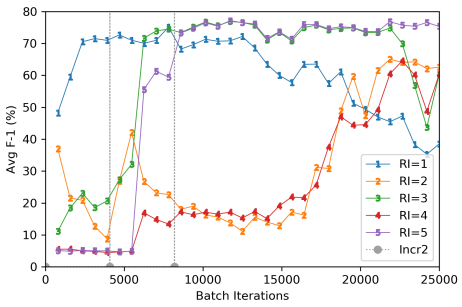
[1] https://github.com/huggingface/pytorch-pretrained-BERT



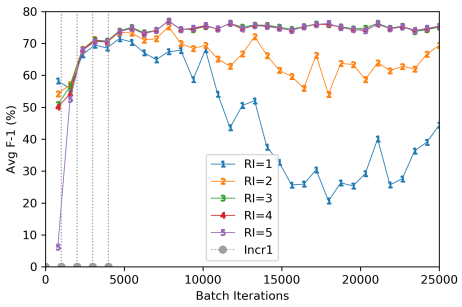
(a) Direct training at target RI w/o warm-up.



(b) 5 RIs w/o random reminder. Training RI incremented by 1 every 4092 batches (1 epoch).



(c) 5 RIs w/o random reminder. Training RI incremented by 2 every 4092 batches (1 epoch).



(d) 5 RIs w/ random reminder. Training RI incremented by 1 every 1000 batches, with a 10% chance ± 1, capped by 1 and 5 RIs.

Figure 3: Training strategies with progressive warm-up and random reminder.

An adaptor layer, consisting of a linear layer followed by layer normalization, converts the concatenated outputs of BERT into a 320-sized vector per token. For the conditional layer, a single layer Transformer encoder is used, with the intermediate size 1280 and 4 attention heads.

The output layer starts with input size 320 to 320, followed by a ReLU and a dropout layer, and then another linear layer from size 320 to the final output logits. The output layer for HA, since it is a sequence-level classification, an attention layer with 8 attention heads is used.

The dropout probability for BERT is set to 0.1. Other dropout probabilities in the conditional layer and the output layer are set to 0.2. The balancing parameter λ between subtasks is set to 0.2.

4.5 Training Parameters

We use the variant ADAM introduced in [9] as the optimizer with the learning rate 5×10^{-5} and use the first half of the epoch for the warm-up process. Batch size is set to 32 and gradient norm is clipped at 1.0. All models are trained for 6 epochs and the model with the best F-1 score is kept. If there are multiple refinement iterations, the F-1 score from the last iteration is the one for comparison.

4.6 Training Strategies for Iterative Layers

Since we only have one set of subtask labels but multiple refinement iterations (RIs) to satisfy, we propose to train the model with progressive RIs and add randomness to the number of RIs to reminder the model what has learned before. Figure 3 showcases these experiments.

4.6.1 Progressive warm-up

Figure 3a shows the curves for two model training events: one with RI $= 1$, which is essentially a MTL case, and the other with RI $= 3$. It can be seen that going straight to multiple RIs becomes

Table 1: Model Performances

Experiment	TL	HA	It.	EM	EM_{Lift}	F1	$F1_{Lift}$	AvNA	TL	HA
HF BERT				72.05		75.55		79.83		
MTL			1	73.99	0.0%	77.19	0.0%	80.70		
MTL + TL	✔		1	73.03	-1.3%	76.73	-0.6%	80.75	97.34	
MTL + TL, HA	✔	✔	1	73.87	-0.2%	76.50	-0.9%	79.70	97.21	87.65
CIR			3	74.09	0.0%	77.05	0.0%	80.88		
CIR + TL	✔		3	73.86	-0.3%	77.28	0.3%	81.01	96.31	
CIR + TL, HA	✔	✔	3	75.26	1.6%	77.76	0.9%	80.95	96.56	89.33
L MTL			1	75.86	0.0%	79.05	0.0%	82.53		
L CIR + TL, HA	✔	✔	3	77.41	**2.0%**	80.31	**1.6%**	83.42	97.24	90.73
Score Diff Thres	✔	✔	3	77.61	2.3%	80.39	1.7%	83.30		
HA Threshold	✔	✔	3	77.76	**2.5%**	80.65	**2.0%**	83.53		
Test Subm. (yo)	✔	✔	3	76.57		79.59				

numerically challenging because the conditional layer is initialized randomly. As an alternative, we choose to start the training with RI $= 1$ to prepare the conditional layer for more RIs later and then progressively increase the number of training RIs until the target RIs. This is shown in Figure 3b. Note that a model is trained progressively with an increasing training RI, but a model can be used to infer with arbitrary numbers of RIs during prediction. For instance, in Figure 3b, the model is always used to predict for 5 RIs, each of which generates one average F-1 score, hence, 5 curves throughput all batch iterations. But the training RI is increased by 1 only after each epoch, as depicted in gray dotted lines.

The increment of the number of training RIs needs to be gradual. Figure 3c shows an experiment when the number of training RIs is increased by 2 at the end of every epoch, i.e., supervision with RI $= 1, 3, 5$ but not 2 or 4. Interestingly, the model is then performant only at RI $= 1, 3, 5$ and the performance for RI $= 2, 4$ improves very slowly. This gives the evidence that the model must be trained at the number of RIs of interest, without skipping the learning.

4.6.2 Random reminder

By observing the learning curves from Figure 3b and 3c, one can notice that once the particular period of a training RI is done, the model starts to "forget" and no longer performs well after a while. This is especially apparent after 12k batch iterations in Figure 3b, the average F-1 scores for each RI takes turns to drop in the reversed order of how they learn.

We propose to mitigate this issue by randomly training the model at different numbers of training RIs. For each batch, we assign a 10% chance to increase the current RI by 1 (capped at the maximum target RI) and another 10% chance to decrease the RI by 1 (capped at 1). This leaves a 80% chance to stay with the schedule RI to train the model. The result is shown in Figure 3d. In this case, we increase RI every 1000 batches with the random reminder specified above. The model is then able to keep the memory and perform well for $RI = 3, 4, 5$, but still degrades for $RI = 1, 2$. One possible way to improve this is to make the random reminder not only remind the RIs close to the scheduled training RIs, but also can cover lower numbers of RIs to prevent this from happening.

4.7 Multitask Learning

When the target number of refinement iteration is exactly 1, there is no feedback loop for the conditional refinement network to learn from previous predictions. In this case, the model with Conditional Iterative Refinement (CIR) degenerates to a regular MultiTask Learning (MTL) model. The 2nd row sets of Table 1 summarizes the performances. As discussed earlier, MTL models (with hard parameter sharing) may not benefit from using multiple subtasks. In fact, we observed 0.6% and 0.9% drops compared to only the start and end position prediction task in the average F-1 scores, with TL and with TL and HA, respectively.

4.8 Conditional Iterative Refinement

For the case of CIR, we choose $r = 3$, again, with the main task as the reference, and with TL and with TL and HA as the test cases. We observed 0.3% and 0.9% lifts for each case. These experiment results coincide with our hypothesis that CIR effectively helps the model learn from previous predictions, potentially a better method for sharing knowledge between subtasks.

4.9 Inference Strategies

Besides training, the predictions from subtasks can also be useful for inference. Since we make HA as a standalone subtask, we can utilize its has-an-answer classification and jointly make a decision with the dummy token approach from the start and end position prediction.

In this experiment, we use the CIR with large uncased BERT, TL, and HA (the row "L CIR + TL, HA" in Table 1) to generate logits from the development set. We would like to use two different methods to determine whether there is an answer. The first method is to use the dummy approach alone: if the product of the start and the end probabilities for the dummy token is larger than the best predicted answer span by a margin, then there is no answer. By tuning the margin, we gain marginally improvement on the average F-1 score from a lift 1.6% to 1.7% (the row "Score Diff Thres"). The second method is to ignore the dummy token entirely, we instead set a threshold for the HA score to determine if there is an answer. In this case, we gain a lift from 1.6% to 2.0%, as shown in the row "HA Threshold."

5 Analysis

We show some qualitative examples of how each iteration refines and diversifies predicted answers. The predicted answers are generated from the model "CIR + TL, HA" in Table 1, with F-1 scores 76.63, 77.75, 77.76 for each refinement iteration. The predicted answers are listed from iteration 1 to 3, in order.

- Context: Context 1 in Appendix
- Question: When was the French version of the word Norman first recorded?
- Answer: *no answer*
- RIs: 9th century, 9th century, *no answer*

At the 3rd RI, the model finds out there is no answer.

- Context: Context 2 in Appendix
- Question: What helped spread Protestantism in France?
- Answer: [The availability of] the Bible in vernacular languages
- RIs: The availability of the Bible in vernacular languages, Bible in vernacular languages, The availability of the Bible in vernacular languages

This is more of a cosmetic change of words. In other words, if there is a predicted answer span, conditional iterative refinement tends to strongly correlate with each iteration.

- Context: Context 3 in Appendix
- Question: What type of musical instruments did the Yuan bring to China?
- Answer: Western
- RIs: Western musical instruments, Western musical instruments, Western

Again, the predicted answers are highly correlated. The refinement slightly changes the span boundary, oftentimes toward improving the metrics.

By inspecting predictions from each refinement iteration, it seems that the model struggles most with whether there is an answer or not, as the model is almost certain about whether the answer span located.

6 Conclusion

We propose a conditional iterative refinement mechanism that has shown a better knowledge sharing capacity than the traditional multitask learning with hard parameter sharing. For the refining schema to be trained effectively, we propose a novel training method that helps iterative layer to converge. With the proposed conditional iterative refinement, we improve the average F-1 score by 2.0% and the exact match score by 2.5%.

7 Future works

The proposed conditional refinement network can be potentially ubiquitously applicable. It is possible that it also works with traditional LSTM types of networks and possibly also works with simple fully connected nets, as long as there are multiple subtasks available. There is still room for improving the training method to make the refinement scheduling and the reminder more adaptive and intelligent to capture weak refinement iterations. Since there are multiple refinement iterations, more advanced inference methods can also be explored to have better synergy between subtasks and refinement iterations.

References

[1] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics.

[2] Hsin-Yuan Huang, Chenguang Zhu, Yelong Shen, and Weizhu Chen. FusionNet: Fusing via Fully-Aware Attention with Application to Machine Comprehension. *arXiv:1711.07341 [cs]*, November 2017.

[3] Natural Language Computing Group. R-NET: Machine Reading Comprehension with Self-Matching Networks. *Microsoft Research*, May 2017.

[4] Wenhui Wang, Nan Yang, Furu Wei, Baobao Chang, and Ming Zhou. Gated Self-Matching Networks for Reading Comprehension and Question Answering. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 189–198. Association for Computational Linguistics, 2017.

[5] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. Bidirectional Attention Flow for Machine Comprehension. *arXiv:1611.01603 [cs]*, November 2016.

[6] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V. Le. QANet: Combining Local Convolution with Global Self-Attention for Reading Comprehension. *arXiv:1804.09541 [cs]*, April 2018.

[7] Yiming Cui, Zhipeng Chen, Si Wei, Shijin Wang, Ting Liu, and Guoping Hu. Attention-over-Attention Neural Networks for Reading Comprehension. *arXiv:1607.04423 [cs]*, pages 593–602, 2017.

[8] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, October 2018.

[10] Fu Sun, Linyang Li, Xipeng Qiu, and Yang Liu. U-Net: Machine Reading Comprehension with Unanswerable Questions. *arXiv:1810.06638 [cs]*, October 2018.

[11] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The Natural Language Decathlon: Multitask Learning as Question Answering. *arXiv:1806.08730 [cs, stat]*, June 2018.

[12] Jason Lee, Elman Mansimov, and Kyunghyun Cho. Deterministic Non-Autoregressive Neural Sequence Modeling by Iterative Refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182. Association for Computational Linguistics, 2018.

[13] Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. Non-Autoregressive Neural Machine Translation. *arXiv:1711.02281 [cs]*, November 2017.

[14] Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. Blockwise Parallel Decoding for Deep Autoregressive Models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10086–10095. Curran Associates, Inc., 2018.

[15] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML*, 2008.

[16] Guillaume Alain and Yoshua Bengio. What Regularized Auto-encoders Learn from the Data-generating Distribution. *J. Mach. Learn. Res.*, 15(1):3563–3593, January 2014.

[17] Richard Caruana. Multitask Learning: A Knowledge-Based Source of Inductive Bias. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 41–48. Morgan Kaufmann, 1993.

[18] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-Adaptive Feature Sharing in Multi-Task Networks with Applications in Person Attribute Classification. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1131–1140, Honolulu, HI, July 2017. IEEE.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, June 2017.

Appendix

Sample context paragraphs

Context 1

The English name "Normans" comes from the French words Normans/Normanz, plural of Normant, modern French normand, which is itself borrowed from Old Low Franconian Nortmann "Northman" or directly from Old Norse Norðmaðr, Latinized variously as Nortmannus, Normannus, or Nordmannus (recorded in Medieval Latin, 9th century) to mean "Norseman, Viking".

Context 2

The availability of the Bible in vernacular languages was important to the spread of the Protestant movement and development of the Reformed church in France. The country had a long history of struggles with the papacy by the time the Protestant Reformation finally arrived. Around 1294, a French version of the Scriptures was prepared by the Roman Catholic priest, Guyard de Moulin. A two-volume illustrated folio paraphrase version based on his manuscript, by Jean de Rély, was printed in Paris in 1487.

Context 2

Western musical instruments were introduced to enrich Chinese performing arts. From this period dates the conversion to Islam, by Muslims of Central Asia, of growing numbers of Chinese in the northwest and southwest. Nestorianism and Roman Catholicism also enjoyed a period of toleration. Buddhism (especially Tibetan Buddhism) flourished, although Taoism endured certain persecutions in favor of Buddhism from the Yuan government. Confucian governmental practices and examinations based on the Classics, which had fallen into disuse in north China during the period of disunity, were reinstated by the Yuan court, probably in the hope of maintaining order over Han society. Advances were realized in the fields of travel literature, cartography, geography, and scientific education.

BERT partial finetuning

Table 2 summarizes an assessment of BERT finetuning and its capacity in a feature-based approach as suggested in the original paper [9]. For SQuAD 2.0, it seems for BERT, finetuning a large number of layers is still essential. In this case, the advantage of using BERT as a pre-trained model is for its better initialization than a out-of-box model that is ready for outstanding few-shot performance.

Table 2: Partial finetuning of Huggingface's Bert for question answering

Model	EM	F1	AvNA
base uncased tuned output only	52.17	52.17	52.12
base uncased tuned last 4 layers	68.10	72.01	77.15
base uncased tuned last 12 layers	72.05	75.55	79.83
large uncased tuned all layers	75.57	78.59	81.92