Multi-Task Deep Neural Networks for Generalized Text Understanding

George He
Stanford
georgehe@stanford.edu

Abstract

In this paper, we present a Multi-Task Deep Neural Network trained across different natural language understanding tasks for learning a common representation.
The MtDNN explored in this paper modifies the model proposed in Liu et al.[18] applied on a different set of natural language tasks. We utilize a baseline pre-trained bidirectional transformer language model (BERT)[4], and demonstrate improvements in the SQuAD 2.0 question answering challenge through incorporating a more generalized model that is able to train across different text understanding tasks, achieving a final validation EM score of 77.853 and F1 score of 81.053.

1 Introduction

Creating a vector-space representation of words and sentences has been fundamental to many natural language understanding tasks. By incorporating popular approaches such as language model pre-training and multi-task learning, models and quickly make use of existing data and obtain well-conditioned, initial starting weights for a variety of natural language understanding problems. In this paper we combine the strengths of both pre-training and multi-task learning approaches by leveraging a new multi-task deep neural network.

Some of the most prominent language model pretraining examples today are ELMo[11] and BERT [4]. These are neural network language models trained on large corpuses of text data, and have improved the state-of-the art performance across the board for natural language understanding tasks. For this paper, we will focus on benefits observed when MtDNN architecture is applied on top of BERT, which is based on a multi-layer bidirectional Transformer[2]. BERT is trained on plain text for masked word prediction and next sentence prediction tasks, and has generally been applied to specific natural language understanding tasks by fine-tuning a final output layer for each task after initiating weights in the pre-trained BERT layers.

Since the underlying language is shared, it makes sense that models trained on natural language understanding tasks share common characteristics that are transferable in one context to another. Multi-task learning mimics the application of the knowledge learned from previous, tangential tasks to help learn a new task more quickly and efficiently. We believe the this improves conventional deep learning models in two ways:

- Diverse pools of data and task objectives lead to regularization through alleviating overfitting to a specific task, thus making the learned representations universal across tasks.
- Supervised learning of deep networks requires large amounts of task-specific labeled data, which is not easily acquired or always available. However, by aggregating related tasks, as in multi-task models, the pool of trainable data is greatly expanded.

By combining the pre-trained weights of BERT with a MtDNN model for an improved training routine that can learn using shared representation layers, we hope to learn more generalized, and better contextualized language models.

2 Related Work

2.1 Pretrained Embeddings

BERT [4] is based on a multi-layer bidirectional Transformer. The publicly released models have been trained on plain text for masked word prediction and next sentence prediction tasks. To apply a pre-trained model to specific NLU tasks, it is often necessary to fine-tune, for each task, the model with additional task-specific layers using task-specific training data.

2.2 Multitask Learning

Multi-task learning[3] as been applied across a plethora of different learning tasks in recent years as availability of large datasets tangentially related to one-other have been publicized. It has been shown to enable zero-shot translation[17], and can be coupled with sequence-to-sequence architectures to multitask across translation, parsing, and image captioning tasks[9] using varying numbers of encoders and decoders. Liu et al [18] recently demonstrated the success of applying multi-task learning to train a modified BERT-based model to obtain state of the art performance on a number of natural language understanding challenges such as General Language Understanding Evaluation (GLUE) benchmark at 82.2%. This strategy has shown promise to leverage different, tangential language datasets in improving performance across the board for models with a high number of parameters.

McCann et al.[12] present a challenge that spans ten tasks, utilizing ten different datasets. Various training strategies are deployed and evaluated, including round-robin batch-level training strategy to jointly train on the full decaNLP datasets. McCann et al. note that some tasks require more iterations to converge in the single-task setting, which suggests that certain tasks are more difficult for the model to learn. As a result, it is necessary to experiment with strategies more complicated than round-robin such as anti-curriculum and curriculum strategies.

2.3 Catastrophic Interference

As networks learn using multiple objectives, one major concern is catastrophic interference between different objectives. Notable efforts in mitigating catastrophic forgetting include creating penalty terms for parameter norms when training on new tasks [8], packing new tasks into already trained networks[10], penalizing the norm of the difference between parameters for previous tasks during parameter updates[6], and determining task-specific paths through networks[5].

3 Approach

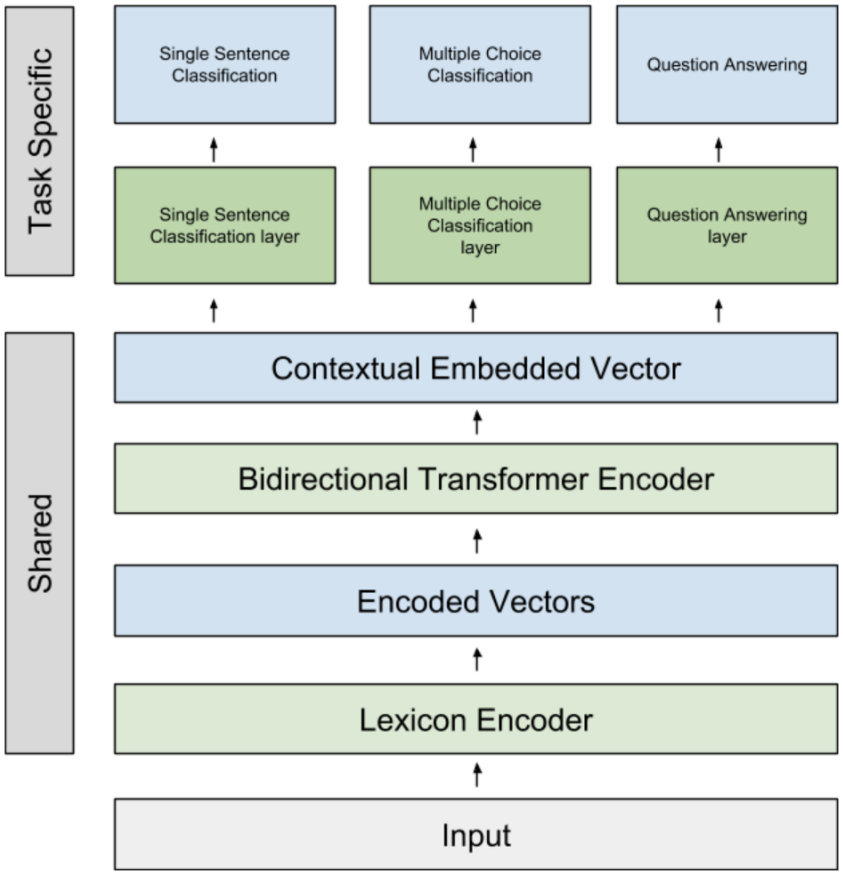
3.1 Model Architecture

The model architecture (Figure 1) contains of a series of shared text encoding layers consisting of a lexicon encoder and a contextual transformer encoder, which are used and updated across all tasks. We use the huggingface[1] pytorch implementation of BERT as a baseline model for the shared text encoding layers. We diverge from the MtDNN model as described by Liu et al[18] by removing an additional Stochastic Answering Network layer across all training tasks added in the previous paper, since we aim to compare the effectiveness of vanilla fine-tuning tasks as published in BERT to multi-task task training. The final output layers are split per-task, and are consequently independently tuned per task. We describe the layers in more detail below:

3.1.1 Lexicon Encoder

The lexicon encoder maps X into a sequence of input embedding vectors, one for each token, constructed by summing the corresponding word, segment, and positional embeddings. The input, which is a word sequence specific to different tasks, is first represented as a sequence of embedding vectors as done in BERT[4]. A vector is produced for each input token, with special handling of the input in the case of SQuAD and other multi-sentence datasets through the pre-pending a *[CL]* token to all inputs and separating sentences with *[SEP]* tokens. This is explained more in detail in the implementation section.

Figure 1: MtDNN model architecture



3.1.2 Contextual Transformer Encoder

The bidirectional transformer-encoder[2] layer then captures the contextual information for each word to generate a commonly shared contextual embedding of size H. The transformer layer applies a self-attention mechanism which directly models relationships between all words in a sentence, regardless of their respective position. In the MtDNN model, the contextual transformer layer serves as a shared representation among different tasks. This differs then task-specific BERT models, where the contextual transformer layer is adapts to each specific task.

3.1.3 Multiple Choice Classification (MCC)

A girl is going across a set of monkey bars. She
(i) jumps up across the monkey bars.
(ii) struggles onto the bars to grab her head...

The multiple choice classification problem, in which the model is tasked with identifying which passage texts which best complete a sentence, can be modeled by constructing multiple input sequences for each question to be evaluated independently - each input a concatenation of the given sentence and possible continuation. The final output layer is a linear layer V of size $[H \times 1]$ mapping the contextual embedding layer to a size one output: $Output = V \star contextualEmbedding$. The probability of a single choice being selected is represented as the softmax over the different choices:

$$P_i = \frac{e^{V*C_i}}{\Sigma_{j=1}^{n_{choices}} e^{V*C_j}}$$

The loss is computed as the cross entropy loss between the known best answer and the softmax probabilities. That is, given input X,

$$loss = -\Sigma_c 1(X, c) log(P_r(c|X))$$

3.1.4 Sequence Classification (SC)

INVALID: In which way is Sandy very anxious to see if the students will be able to solve the homework problem?
VALID: The book was written by John.

Given a sentence, the model labels it using one of the predefined class labels by a linear mapping through the vector V of size $[H \times n_{labels}]$ from the contextual embedding vector to a vector of size n_{labels}.

The probability of a single label choice being selected is represented as the softmax over the linear output layer:

$$P_i = \frac{e^{V*C_i}}{\Sigma_{j=1}^{n_{labels}} e^{V*C_j}}$$

Similar to the multiple choice classification task above, the loss is computed as the cross entropy loss between the known best label and the softmax probabilities.

3.1.5 Question Answering

Text: The Normans were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France.
Question: In what country is Normandy located?
Answer: France

Given a set of texts, questions, and answers, the model aims to produce the correct sequence of words (found in the text) that answer the question. It is possible for there to be no answer, in which case the model should output no answer as well.

The model learns a linear layer V of size $[H \times 2]$ mapping the contextual embedding layer to a size two output: $Output = V \star contextualEmbedding$.

In this way, each location has a start and end probability after the final output layer. Additionally, this model accounts for null possibilities by using the weight assigned to the CLs index as a proxy for the null probability. We say that if the null probability is better than any other non-null probability by a threshold, we assign a "null" value to the output - setting the start and end to 0.

The probability of selecting any specific token as the start or end token is treated as

$$P_i = \frac{e^{V*C_i}}{\Sigma_{j=1}^{n_{tokens}} e^{V*C_j}}$$

The loss is computed as the summed cross entropy loss between the selected start and end.

3.2 Baseline

For this report, we will be comparing against a model solely trained on a single task (SQuAD) using the publicly available BERT-BASE and BERT-LARGE[4] models as an initialization point for the embedding and transformer layers. Due to resource constraints, we will be conducting all ablation studies and experimentation on BERT-BASE. We will report only baseline and final result scores for BERT-LARGE.

3.3 Training Technique

Rather than round robin or shuffling approaches, which require an epoch to iterate through all data without the ability to pre-determine how to emphasize training for a specific objective, we use a weighted approach.

Learning rate will be maintained between different learning objectives. In order to account for imbalances between the importance of the datasets, we experiment with a process we define as weighted stochastic batch selection due to a SQuAD-specific objective. Specifically, the probability of a model being drawn for the next training batch is pre-determined by weights assigned to each model. For this paper, we will uniformly draw from all batched datasets, with the exception of a higher weight for the QA (SQuAD 2.0) dataset. For the training sequence we complete an epoch once all QA training datapoints have been sampled for computational tractability.

$$P(B[i] = b_x) = \frac{W(b_x)}{\Sigma_{x'}^{n_{batches}} W(b_{x'}}$$

3.4 Original Implementation

Our original contributions center around developing a new architecture wherein a single model can be trained using different inputs, on top of the Huggingface BERT implementation, and developing a training model to handle task-specific objectives while utilizing multitask data. Specifically, our model handles the specific tasks outlined above.

4 Experiments

All bert-base experiments were performed Tesla P40 GPUs.
All bert-large experiments were performed on Titan X GPU.
Unless otherwise noted, experiment results are reported for bert-base models.

4.1 Datasets

For multiple choice classification (MCS), we use the SWAG[14] dataset, which contains 113k multiple choice questions about a rich spectrum of grounded situations.

For sequence classification(SC), we use the the Corpus of Linguistic Acceptability(CoLA)[15] dataset, which consists of 10657 sentences from 23 linguistics publications, expertly annotated for acceptability (grammaticality). Another dataset we will focus on in this corpus include the Microsoft Research Paraphrase Corpus (MRPC) dataset, which contains 5800 pairs of sentences which have been extracted from news sources on the web, along with human annotations indicating whether each pair captures a paraphrase/semantic equivalence relationship. The Multi-Genre Natural Language Inference(MNLI)[16] corpus is a crowd-sourced collection of 433k sentence pairs annotated with textual entailment information.

For question answering, we use the Stanford Question Answering Dataset (SQuAD) 2.0 dataset [13], which combines existing SQuAD reading comprehension data, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable, with over 50,000 unanswerable questions.

4.2 Evaluation Method

As in Rajpurkar et al.[13], we use the Exact Match Score when applying the model the SQuAD 2.0, which is measures the percentage of predictions that match any one of the ground truth answers exactly. We also use the F1 Score when applying the model the SQuAD 2.0, which measures the average overlap between the prediction and ground truth answer. We treat the prediction and ground truth as bags of tokens, and compute their F1. We take the maximum F1 over all of the ground truth answers for a given question, and then average over all of the questions.

4.3 Implementation Details

Our implementation of MT-DNN is based on the PyTorch[1] implementation of BERT. We used Adamax[7] as our optimizer with weight decay fix, warmup and linear decay of the learning rate. A linear learning rate decay schedule with warm-up over 0.1 was used, unless stated otherwise.

Datasets	EM	F1
QA(0.5)+MCC+SC	72.08	75.224
QA(0.7)+MCC+SC	73.445	76.426
QA(0.8)+MCC+SC	74.58	77.15
QA(0.9)+MCC+SC	73.92	76.47
QA(1.0)	73.544	76.18

Table 1: Varying QA weights: bert-base performance on dev dataset

Ideally, each specific task will have different learning rates, as the knowledge learned from each task not uniform. Learning these weights, however, is a hyperparameter tuning experiment that is not tractable given limited compute power. For simplicity, we apply the same learning rate of $3e^{-5}$ to all experiments, using 2 training epochs. The dropout rate is set to 0.3 unless otherwise specified.

For bert-base experiments, we use batch sizes of 16, using a pretrained bert-base-cased model. Each experiment took between 6 through 12 hours to complete, depending on the weight of the SQuAD model.

For bert-large experiments, we use batch sizes of 16, using a pretrained bert-large-cased model. Each experiment took between 12 through 15 hours to complete, depending on the weight of the SQuAD model. The dropout rate is set to 0.3 unless otherwise specified.

All the texts were tokenized using wordpieces, and were chopped to spans no longer than 384 tokens, with a maximum question length of 64 for question answering datasets.

Question answering inputs are encoded as a single vector , with special handling of the input in the case of SQuAD through the pre-pending a *[CL]* token to all inputs and separating input context, and query sentences with *[SEP]* tokens.

Sequence classification inputs are encoded as a single vector, with handling of the input datasets through the pre-pending a *[CL]* token to all inputs. If an input source consists of multiple-sentences, the sentences are separated with *[SEP]* tokens.

Multiple choice classification inputs are encoded as a single vector for each choice, with handling of the input datasets through the pre-pending a *[CL]* token to all inputs. The linear layer outputs a single value for each choice of a multiple choice problem, then all the outputs corresponding to an instance are passed through a softmax to get the model choice. The input sentence and each choice are separated with *[SEP]* tokens.

4.4 Experiment Details

We experiment with mixing the QA (SQuAD 2.0) dataset with combination of other datasets. We focus on the following data mixture settings for reporting performance

1. (Baseline) Vanilla QA
2. QA + SC
3. QA + MCC
4. QA + SC + MCC

Training will occur in the following manner using minibatch based gradient descent to learn the parameters of our model: In each epoch, a mini-batch is selected randomly from a weighted stochastic process as described above, among all the tasks, and the model is updated according to the computed task-specific objective previously mentioned. This allows the model to optimize the sum of all multi-task objectives. A more detailed outline is provided in the appendix (fig 2).

We experiment with different weighting of the question answering dataset during stochastic batch selection, with uniform weights applied to the remaining datasets. Results from experimenting with different question answering weights are shown in Table 1, and a chart is available in the appendix section (Figure 4)

After determining an optimal QA weight of 0.8, we apply the weight uniformly to other dataset combinations. The results of our observations can be found in Table 2, with QA+MC+SC outperforming

Datasets	EM	F1
QA	73.544	76.18
QA+MCC	73.988	76.754
QA+SC	73.56	76.435
QA+MCC+SC	74.58	77.15

Table 2: bert-base multitask performance on dev dataset

Datasets	EM	F1
QA	77.902	80.831
QA+MCC+SC	78.578	81.573

Table 3: bert-large multitask performance on dev dataset

other training routines, resulting in an improvement over the baseline EM score by 1.04 and the baseline F1 score by 0.97.

It is observed that while weighting the SQuAD dataset too heavily produces diminishing benefits from the multi-task model, weighting the SQuAD dataset minimally can also lead to decreases in performance. It is surprising to find that decreasing the weighting of the SQuAD dataset beyond 0.7 results in a noticeable decrease in performance. It is well known that catastrophic interference in learning[8] can result when training models on multiple objectives, and the choice of optimizer can result in less-than-optimal conditions for improved learning when training in a multi-task setting.

The best observed improvements on the model after applying a multi-task training routing of approximately one point in both EM and F1 scores fall within expectations, as it reflects the observations by Liu et Al.[18], wherein application of the MtDNN training route improved BERT performance in very minute amounts (sub-single digit gains) across most task-specific objectives. This indicates that the MtDNN routine can be applied more generally to architectures with common shared embedding layers, and custom output layers, for improvements across the board. A chart comparing results from different run can be found in the appendix section (Figure 3).

We find the final tuned bert-base model combining QA, SC, and MCC datasets achieved a **EM score of 73.373**, and **F1 score of 76.421** on the validation dataset. We also find the final tuned bert-large model combining QA, SC, and MCC datasets achieved a **EM score of 77.853**, and **F1 score of 81.053** on the validation dataset. Note that this value is lower than the reported SQUAD 2.0 performance of task-specific BERT models likely due to fine-tuning differences.

5 Analysis

Note that analysis of experiment results during ablation studies is conducted in the experiments section as well. All observations are based on bert-base output.

5.1 Improved Answer Selection

Context: One of the first Norman mercenaries to serve as a Byzantine general was Hervé in the 1050s. By then however, there were already Norman mercenaries serving as far away as Trebizond and Georgia. They were based at Malatya and Edessa, under the Byzantine duke of Antioch, Isaac Komnenos. In the 1060s, Robert Crispin led the Normans of Edessa against the Turks. Roussel de Bailleul even tried to carve out an independent state in Asia Minor with support from the local population, but he was stopped by the Byzantine general Alexius Komnenos.
Question: Who ruined Roussel de Bailleul's plans for an independent state?
QA+MCC+SC answer: Alexius Komnenos
QA+SC answer: Alexius Komnenos
QA+MCC answer: Empty
QA answer: Empty

Context: 'Eventually, the Normans merged with the natives, combining languages and traditions. In the course of the Hundred Years' War, the Norman aristocracy often identified themselves as English. The Anglo-Norman language became distinct from the Latin language, something that was the subject of some humour by Geoffrey Chaucer. The Anglo-Norman language was eventually absorbed into the Anglo-Saxon language of their subjects (see Old English) and influenced it, helping (along with the Norse language of the earlier Anglo-Norse settlers and the Latin used by the church) in the development of Middle English. It in turn evolved into Modern English.'
Question: What was the Anglo-Norman language's final form?
QA+MCC+SC answer: Modern English
QA+SC answer: Modern English
QA+MCC answer: Empty
QA answer: Empty

We observe that the QA+MCC+SC trained model performs better at determining recognition in different contexts. Improvements in simple recognition are hypothesized to be brought about by inclusion of the Microsoft Research Paraphrase Corpus (MRPC) dataset, which contains sentences which have been extracted from news sources on the web, along with human annotations indicating whether each pair captures a paraphrase/semantic equivalence relationship, provide strong soft-metrics for question-answering specific output layers to determine good paraphrasing between context, question, and answer. We observe above from the improved answers, that incorporation of the sequence classification datasets improve the predicted answer compared to training routines that do not incorporate the sequence classification training data.

5.2 Grammatical Improvements - Proper Answer Structure

Context: On October 6, 1973, Syria and Egypt, with support from other Arab nations, launched a surprise attack on Israel, on Yom Kippur. This renewal of hostilities in the Arab–Israeli conflict released the underlying economic pressure on oil prices. At the time, Iran was the worldś second-largest oil exporter and a close US ally. Weeks later, the Shah of Iran said in an interview: "Of course [the price of oil] is going to rise... Certainly! And how!... Youv́e [Western nations] increased the price of the wheat you sell us by 300 percent, and the same for sugar and cement... You buy our crude oil and sell it back to us, refined as petrochemicals, at a hundred times the price youv́e paid us... Itś only fair that, from now on, you should pay more for oil. Letś say ten times more."
Question: How many times more did the other nations have to pay for oil after the surprise attack?
QA+MCC+SC answer: ten times more
QA+SC answer: ten times more
QA+MCC answer: ten
QA answer: ten

We observe that the responses provided by the MtDNN model trained on QA+MC+SCC does a much better job at restating the question with proper grammar in the provided context where possible. Notably, incorporation of data from the Corpus of Linguistic Acceptability(CoLA)[15] dataset, which consists of sentences expertly annotated for acceptability (grammatically) likely provided soft-metrics for the question answering specific output layer to improve the grammatical structure of responses, and subsequently, improve the question answering performance score. Above we see number of examples illustrating sentences with improved grammatical context, having been improved from incorpration of sequence classification data.

6 Conclusion

In this project we implemented an end-to-end neural network model to perform reading comprehension on the SQuAD 2.0 dataset. In our approach, we combined several state-of-the-art techniques such as pretrained embeddings[4], transformer encoders[2], and multitask learning [18].

Although ablation tests indicate that multi-task information must be carefully incorporated for a beneficial outcome, we show that when incorporated carefully, application of a common embedding and encoding model on multi-task learning for shared language understanding successfully demonstrated an improvement in performance on a targeted task, in comparison to models that do not incorporate multi-task data and objectives.

7 Acknowledgements

We would like to thank the CS224N teaching staff for providing project guidance. We would also like to thank Microsoft for sponsoring the class and providing Azure credits to perform experiments on.

References

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

[2] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. `https://arxiv.org/abs/1706.03762`, 2017.

[3] Rich Caruana. Multitask learning. *Mach. Learn.*, 28(1):41–75, July 1997.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[5] Chrisantha Fernando, Dylan Banarse, Charles Blundell, Yori Zwols, David Ha, Andrei A. Rusu, Alexander Pritzel, and Daan Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017.

[6] Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. A joint many-task model: Growing a neural network for multiple NLP tasks. *CoRR*, abs/1611.01587, 2016.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[8] James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016.

[9] Minh-Thang Luong, Quoc V. Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. Multi-task sequence to sequence learning. *CoRR*, abs/1511.06114, 2015.

[10] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *CoRR*, abs/1711.05769, 2017.

[11] Mohit Iyyer Matt Gardner Christopher Clark Kenton Lee Luke Zettlemoyer Matthew E. Peters, Mark Neumann. Deep contextualized word representations. `https://arxiv.org/abs/1802.05365`, 2018.

[12] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *CoRR*, abs/1806.08730, 2018.

[13] Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018.

[14] Roy Schwartz Yejin Choi Rowan Zellers, Yonatan Bisk. Swag: A large-scale adversarial dataset for grounded commonsense inference. `https://arxiv.org/abs/1808.05326`, 2018.

[15] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2018.

[16] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.

[17] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[18] Weizhu Chen Jianfeng Gao Xiaodong Liu, Pengcheng He. Multi-task deep neural networks for natural language understanding. `https://arxiv.org/abs/1901.11504`, 2019.

8 Appendix and Supplementary Materials

Figure 2: MtDNN Training Algorithm, with focus on Question Answering

Algorithm 1: Weighted Batch Stochastic Selection

Initialize model parameters θ randomly;
Pre-train all shared layers with known BERT weights (the lexicon encoder and transformer encoder;
Set the max number of epochs and learning rates;
Set the weight distribution W;
Prepare the data for all tasks;
for *t in 1,2,....T* **do**
  Pack the dataset t into mini-batch D_t;
end
for *epoch in 1,2....$epoch_{max}$* **do**
  $C = D.copy()$;
  while *$C_{QuestionAnswering}$ is not empty* **do**
    Select batch type: $BatchType = RandomBatch(Weight)$;
    Pop batch: $b_t = C_{BatchType}.pop()$;
    Compute loss: $Loss = Loss_{BatchType}(\theta, b_t)$;
    Compute Gradient: $Gradient = \nabla(\theta)$;
    Update model: $\theta = \theta - \epsilon\nabla(\theta)$;
  end
end

Figure 3: QA Weights vs Score

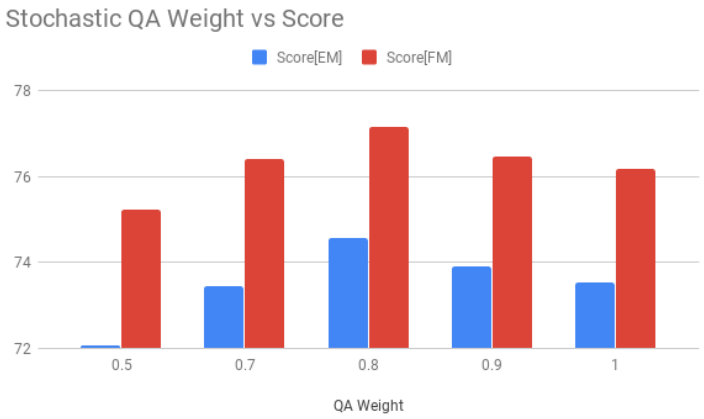


Figure 4: Datasets vs Score