Enhancing Task-Specific Distillation in Small Data Regimes through Language Generation

Husam Quteineh[2,1,*], **Spyridon Samothrakis**[3,1], **and Richard Sutcliffe**[4,1,*]

[1]School of CSEE, University of Essex, UK
[2]Business and Local Government Data Research Centre, University of Essex, UK
[3]Institute for Analytics and Data Science (IADS), University of Essex, UK
[4]School of Information Science and Technology, Northwest University, China
husam.quteineh@essex.ac.uk, ssamot@essex.ac.uk, rsutcl@essex.ac.uk, rsutcl@nwu.edu.cn

Abstract

Large-scale pretrained language models have led to significant improvements in Natural Language Processing. Unfortunately, they come at the cost of high computational and storage requirements that complicate their deployment on low-resource devices. This issue can be addressed by distilling knowledge from larger models to smaller ones through pseudo-labels on task-specific datasets. However, this can be difficult for tasks with very limited data. To overcome this challenge, we present a novel approach where knowledge can be distilled from a teacher model to a student model through the generation of synthetic data. For this to be done, we first fine-tune the teacher and student models, as well as a Natural Language Generation (NLG) model, on the target task dataset. We then let both student and teacher work together to condition the NLG model to generate examples that can enhance the performance of the student. We tested our approach on two data generation methods: a) Targeted generation using the Monte Carlo Tree Search (MCTS) algorithm, and b) A Non-Targeted Text Generation (NTTG) method. We evaluate the effectiveness of our approaches against a baseline that uses the BERT model for data augmentation through random word replacement. By testing this approach on the SST-2, MRPC, YELP-2, DBpedia, and TREC-6 datasets, we consistently witnessed considerable improvements over the word-replacement baseline.

1 Introduction

Transformer-based models have shown wide success over a variety of Natural Language Processing (NLP) tasks. Their ability to scale up to trillions of parameters made it possible to acquire and transfer generalized knowledge from large collections of data to downstream tasks. While these models can lead to significant improvements in performance, the increasing size of their learning parameters results in greater complexity and storage requirements. This can be challenging in real-time applications, especially on devices with limited computational resources (Gou et al., 2021). Hence, reducing the size of these language models without sacrificing too much of their performance has become the focus of many works in Knowledge Distillation (KD). Instead of optimizing compressed models for hard-labeled data, Hinton et al. (2015) proposed to train a smaller model (the student) with the task of predicting the probability distribution outputs (soft labels) from a larger model (the teacher). This approach has been shown to produce comparable results between the student and teacher models, but usually relies on a large enough dataset through which knowledge can be transferred. To help improve the student's learning in KD, large unlabeled datasets are required (Tang et al., 2019). Although unlabeled data is cheaper to obtain and is widespread when compared to labeled data, it may not be available for every task and application. We therefore propose to generate synthetic examples that can be used to transfer knowledge to the student in downstream tasks.

To overcome the challenges that come with unavailability of large unlabeled datasets required for the distillation process, we build a data generation framework where the Monte Carlo Tree Search (MCTS) algorithm is applied to help generate examples that, if added to the student's training data, will increase its performance. By taking the difference between the student's and the teacher's

[*] Corresponding author

uncertainty for a generated example, we are able to force the language generation model to create examples that can be pseudo-labeled by the teacher with higher confidence than its student. We make the assumption that the wider the gap between the student's and teacher's uncertainty for a particular example, the more likely it is that this example is correctly pseudo-labeled by the teacher and the less likely that it is known to the student model. By training the student on the generated data with the teacher's pseudo labels, it is able to improve its performance by mimicking its teacher's behavior. To generate examples that meet this condition, we take advantage of MCTS's tendency to search for paths that maximize the reward value, hence, the uncertainty gap. The intuition here is that the larger the positive difference in the uncertainty, the greater the contradiction is between the student and its teacher, and the more likely that the generated example is important for the student's learning stage. We also show that strong results can still be obtained with a random generation approach that does not optimize for a reward function during the generation process. Instead, it first generates data, then selects samples that meet the conditions that are set in the reward function. The contributions of the paper are:

- We propose Monte Carlo Text Generation (MCTG), a novel method in KD which uses MCTS to generate synthetic examples.
- We present Non-Targeted Text Generation (NTTG), in which data is first generated with top-k sampling, then filtered on the conditions of the reward function.
- We show that even when starting with a few examples per label, we can massively improve the student's performance.

The remainder of the paper is structured as follows: Section 2 provides a background and an overview of related literature. Section 3 describes the proposed approach. Section 4 presents the experiments which were carried out. Section 5 gives conclusions and plans for future work.

2 Background

In the pursuit of improving performance for natural language processing, pretraining large-scale models on increasing amounts of unlabeled data has become a common approach (Devlin et al., 2018; Peters et al., 2018; Yang et al., 2019). By leveraging the knowledge gained from the pretraining step, these models have shown impressive performance on many downstream text tasks, e.g. GLUE and SuperGLUE benchmarks (Wang et al., 2018, 2019). However, such improvements are accompanied by an increasing number of learning parameters, creating a need for more computational and storage requirements. To alleviate the aforementioned complexity issues, many have suggested approaches for efficient training through model optimization e.g. removal of inefficient or redundant parameters (Lan et al., 2020; Sajjad et al., 2020), and knowledge distillation (Gou et al., 2021; Sun et al., 2019, 2020; Sanh et al., 2019). In knowledge distillation, the unlabeled data plays an intermediary role, which allows the teacher to transfer its knowledge through its predictions. When this data lacks the components for a meaningful knowledge transfer, e.g. limitations in size or textual variations, augmentation techniques can be applied to create additional examples. For instance, Tang et al. (2019); Jiao et al. (2019) apply task-agnostic heuristics like word replacements, to improve distillation on downstream tasks. The concept of augmenting training examples has been successfully shown to improve training in computer vision (Shorten and Khoshgoftaar, 2019), and has been gaining traction in the NLP domain as well. This includes word manipulations such as the deletion, insertion, or addition of random words in text (Wei and Zou, 2019), paraphrasing or back-translation (Sennrich et al., 2015), and most recently the application of language models to predict alternative words (Kobayashi, 2018).

In this work, instead of relying on the above augmentation techniques, we propose to improve knowledge transfer by involving the student and the teacher in the creation of useful examples. We achieve this through a framework that uses an NLG model to create examples that are deemed useful for knowledge distillation. The steps taken to determine the usefulness of an example are summarized in Figure 1. In MCTG, we achieve this by exploring MCTS's ability for finding optimal solutions which are rewarded by the usefulness of the examples they represent, as explained in section 3.2.

2.1 Knowledge Distillation (KD)

Knowledge distillation is typically aimed at training a student model to mimic the behavior of a larger teacher model. The student can either have

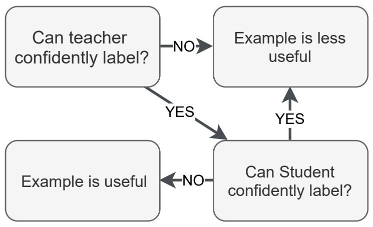
Can teacher confidently label?
NO
Example is less useful
YES
YES
Example is useful
NO
Can Student confidently label?

Figure 1: An example is deemed more useful if the teacher can confidently label it, but not the student.

the same architecture as its teacher or be completely different from it, but in either case, it usually has fewer learning parameters. Hinton et al. (2015) achieve knowledge distillation by training the student model on the softmax probabilities of the teacher model. Other KD approaches have also been proposed, which include the distillation of the activations or weights of the intermediate layers (Romero et al., 2014; Tarvainen and Valpola, 2017; Yim et al., 2017; Heo et al., 2019; Cho and Hariharan, 2019). In contrast to much work in KD, we deny both the teacher and its student access to the full training datasets and only train them on a small sample of seed data. Hence, our approach does not depend on pre-existing large datasets for distilling knowledge. Instead, our work focuses on very small data settings. Hence, we make the proposition that fine-tuning compact models on small datasets can be aided by the participation of larger models in a) generating additional training examples, and b) finding informative examples while providing pseudo labels.

2.2 Language Models

Traditional context-independent word vectors like GLOVE (Pennington et al., 2014) and Word2Vec (Mikolov et al., 2013) were popular choices for initializing embedding layers for task-specific networks. Later works focused on contextualizing representations by leveraging recurrent neural networks (Peters et al., 2018; Howard and Ruder, 2018); most recently, the fine-tuning of pretrained transformer-based models (Vaswani et al., 2017) like BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019) and GPT-2 (Radford et al., 2019), has become a common approach for domain-specific tasks. In our experiments, we generate language with GPT-2, a unidirectional language model, pre-trained on large textual datasets with a probabilistic function to estimate the probability distribution over the vocabulary for a given context. For a sequence of tokens $t_1, t_2, t_3, ..., t_n$, the joint probability can be modeled as:

$$p(t) = \prod_{i=1}^{i=n} p(t_{(i)}|t_{(1)}, \ldots, t_{(i-1)}) \qquad (1)$$

The conditional probability of a token $p(t_i|t_{1:i\_1})$ can be estimated by the probability distribution over the model's vocabulary given a context $t_{1:i\_1}$. Thus, we can generate candidates for every next token with top-k sampling (Fan et al., 2018). When a token is selected, and the process is repeated enough times, a properly trained model can generate a meaningful sequence of text. Even though we restrict our approach to small datasets, our experiments show that GPT-2 is still able to generate relevant examples.

2.3 Monte Carlo Tree Search (MCTS)

MCTS has been widely applied to gaming theory (Kocsis and Szepesvári, 2006; Browne et al., 2012). Its ability to solve decision-making problems in games with large combinatorial search spaces (Silver et al., 2016; Arneson et al., 2010; Chung et al., 2005), has made its application appealing even for non-gaming problems as well (Nguyen et al., 2016; Edelkamp et al., 2016). In our previous work, we showed that MCTS can also be applied to create synthetic data for text classification tasks (Quteineh et al., 2020).

MCTS incrementally constructs a tree as it searches for possible solutions that satisfy the conditions set by its reward function. In computer games, paths that lead to winning states are more likely to have higher reward values than paths that lead to losing states. While any of the winning paths could be equally desirable, some paths could have a higher probability of reaching a winning state than others. By keeping track of the number of visits MCTS makes with every path it takes, we can safely assume that winning paths with a higher number of visits are more likely to reach a winning state. However, if the path selection criterion focuses only on maximizing the reward value, it can repeatedly revisit the same paths while failing to discover new ones. To account for this, a selection policy must balance between exploration of new paths and exploitation of already visited paths. A common policy that can achieve this balance is the Upper Confidence Bound (UCB), proposed by Auer et al. (2002) for solving the multi-armed bandit problems, as shown in equation 2:

$$UCB = \frac{W_i}{S_i} + C\sqrt{\frac{2 \times lnS_p}{S_i}} \quad (2)$$

Where for a given state i, W_i represents the number of paths leading to a winning state, and S_i records the total number of paths from i. The first part of the equation, $\frac{W_i}{S_i}$, favors paths that have on average led to a winning state, whereas the second part of the UCB policy, $C\sqrt{\frac{2 \times lnS_p}{S_i}}$, favors unexplored paths. S_p is the total number of paths taken from the parent node, and C is an exploration hyperparameter. A higher C would increase exploration. UCB combined with MCTS is commonly known as the Upper Confidence Bound for Trees (UCT) (Browne et al., 2012). The final MCTS algorithm can be divided into four main stages:

1. Selection: Starting from a root node S_r, the UCB function from eq. 2 is recursively applied to select the next node to visit, until an unexpanded node is reached.
2. Expansion: A non-terminal leaf node is expanded by adding its immediate children. This corresponds to all the immediate actions that are possible from that state.
3. Simulation: From the current state, a random path ending with a terminal state is generated.
4. Backpropagation: Once a terminal node is reached, statistics including the reward value and the number of visits are backpropagated to the nodes of the current path.

Typically, MCTS runs until a predefined criterion is satisfied, e.g. a user-specified time or a maximum number of iterations. We adapt MCTS so that each node represents a token generated by GPT-2, where the possible actions k from a particular node S_i are from a top-k sampling process. Each full path represents a complete text example that is rewarded by equation 6.

3 Approach

In this work, we propose the Monte Carlo Text Generation (MCTG) method (section 3.2), alongside the Non-Targeted Text Generation (NTTG) method (section 3.4). In MCTG, we combine MCTS, a language generation model, a teacher, and a student classifier to create synthetic examples that can enhance the performance of the student in a KD

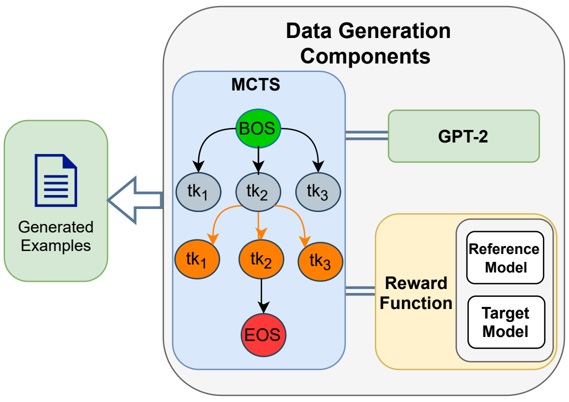

Figure 2: MCTS builds a tree from token sequences generated by GPT-2. Meanwhile, the teacher and student models work together to reward for completed paths.

setting. Here, the language model is conditioned to generate examples for which the predictions of the teacher and its student are as divergent as possible. The main components of our framework, as shown in Figure 2, include the language generation model (GPT-2), a teacher model, a student model, and the MCTS algorithm. Below we discuss the role of each component.

3.1 Language Generation Model

Because the search tree is constructed by traversing from top to bottom, a unidirectional generative model can take tokens of parent nodes as input to predict candidates for the next node. This unidirectional behavior makes GPT-2 a good choice for our experiments[1]. To generate relevant data, GPT-2 is first fine-tuned on the initial training dataset. We prepare the data by first dropping the target labels, then merging the text examples, split by <|endoftext|>. The fine-tuned GTP-2 is then used to generate a token for each node, tk, in the constructed tree, as shown in Figure 2.

3.2 Monte Carlo Text Generation (MCTG)

We refer to the application of MCTS for text generation as MCTG. Starting from a root node, represented by a special token, $< |endoftext| >$, we sample the top k most probable tokens that come next in sequence. Having only started from a single root node, the tree is first expanded by adding the top k tokens as immediate children, making the depth of the tree equal to 2. Since at this stage all child nodes have equal weight, any one of them is selected. The next step is to start a simulation by

[1]We use Huggingface https://huggingface.co/

first concatenating the token of the selected node with those of its ancestor nodes. The resulting text is then passed to GPT-2 to obtain the probability distribution over the vocabulary, where the top k tokens are sampled. Given that this process takes place in the simulation stage, the UCB selection strategy is not applied; instead, we select a random token from the non-uniform distribution of the k tokens. In a standard MCTS implementation, the path that is navigated in the simulation phase is not necessarily recorded, but rather the statistics of it are, e.g. result and number of visits. However, to account for computational costs, we also track the generated text at the end of every simulation. It is important to note that the tokens generated during a simulation are not added as nodes to the tree, but are recorded separately. In this way we guarantee that while the growth of the tree is not affected during a simulation, we nevertheless retain the generated text examples with positive rewards. After a number of iterations, the statistics of the tree nodes will have changed, allowing higher impact of the UCB policy (e.q 2) in searching for paths that lead to examples with the higher reward values.

3.3 Reward Function

This component plays a key role in our approach as it dictates the usefulness of the generated data. In a student-teacher KD application, the aim is to find examples that the teacher model can label with higher certainty than its student. As entropy measures the uncertainty of a model's prediction for a particular example, the higher the entropy, the lower the confidence of the classifier in its prediction. This motivates us to generate examples that can be predicted with low entropy by the teacher, but high entropy by the student. Hence, when the difference in entropy between the two models is increased, the more important the generated example becomes for training the student. Given a generated text example x_u, the predicted probabilities from a model m are outputs of its $softmax$ layer:

$$P_m(y) = softmax(f(x_u)) \tag{3}$$

The entropy is thus:

$$H_n(P_m) = -\sum_{i=1}^{n} p_i \log_b p_i \cdot \frac{1}{\log_b n} \tag{4}$$

where the predicted probabilities $P_m = \{p_i; i = 1, ..., n\}$ for n labels. We take the difference in entropy between the student s, and the teacher t:

$$\Delta\mathrm{ent} = \mathrm{ent}_s - \mathrm{ent}_t \tag{5}$$

The teacher's confidence and the student's lack of confidence in labeling an example are reflected in $\Delta\mathrm{ent}$. For predictions where the teacher's confidence is at its highest, and the student confidence at its lowest, $\Delta\mathrm{ent}$ is maximized. Hence, examples with high $\Delta\mathrm{ent}$ are more useful for distilling knowledge to the student model. For this reason, we only consider examples where $\Delta\mathrm{ent}$ is positive. By finding paths that maximize $\Delta\mathrm{ent}$, GPT-2 is conditioned to generate examples that can be predicted with the lowest uncertainty by model t, but with the highest possible uncertainty by model s. Here we make the following assumptions: a) Examples that maximize the gap between entropy values are those most informative to the student model, yet can be confidently labeled by the teacher model; b) Training the student on informative examples can increase its performance. Since the objective is to find solutions that maximize the reward value, each generated path is rewarded by $\Delta\mathrm{ent}$. To further optimize the search process, we add the following refinements to the reward value.

$$reward = \begin{cases} 0, & \text{if \#tokens} < x, x \in \mathbb{Z}_{\geq 0} \\ 0, & \text{if } \Delta\mathrm{ent} < 0 \\ -1, & \text{if task specific condition} \\ \Delta\mathrm{ent}, & \text{otherwise} \end{cases} \tag{6}$$

Condition 1 in equation 6 is a heuristic that penalizes examples below a user-defined minimum number of tokens. Condition 2 minimizes the penalty for examples where the student model is more certain in its prediction than its teacher. Condition 3 eliminates cases based on a task-specific condition, see TREC-6 configuration in section 4.4. Finally, the fourth condition results in a positive reward when the teacher is more certain than the student.

3.4 Non-Targeted Text Generation (NTTG)

In NTTG, examples are generated without conditioning GPT-2 on the predictions of the student and the teacher classifiers. Instead, examples are generated purely on the probability distributions for the candidate tokens from GPT-2, then filtered on the conditions from equation 6. As in the MCTS simulation phase, examples are generated by applying top-k sampling on the outputs of GPT-2. In top-k sampling, the probability mass is redistributed over the top k most probable choices. At each timestep, k candidate tokens are sampled based on the previously generated tokens. A token is then randomly selected from the k most likely candidates. A sequence is completed when a symbol indicating the end-of-sequence is selected, or when a user defined maximum sequence length is reached. Due to the randomness in selecting the next token, a different sequence can be produced whenever the generation process is repeated. Once enough examples are generated, the result data is cleaned by removing duplicates and short sequences, e.g. less than 3 tokens. Next, The entropy, e.q. 4, of both the teacher and student models is then computed for each remaining sample. We denote the student's entropy by ent_s and the teacher's entropy by ent_t. With equation 5, for each sample, we compute the difference of entropy between the teacher and its student, $\Delta\mathrm{ent}$. We then apply equation 6, and select the examples with $\Delta\mathrm{ent} > 0$.

4 Experiments

4.1 Baseline: Conditional BERT (C-BERT)

C-BERT baseline augments the training data by applying the 12-layer pretrained BERT model to predict a substitute word for a masked token (Wu et al., 2019). Each token in an input has a 10% probability of being masked, i.e., replaced by a BERT prediction (Kobayashi, 2018). Similar to our GPT-2 generation for the MCTG and NTTG experiments, the replacement token is selected from the top-20 tokens given BERT's probabilities.

4.2 Classification Models

We based our experiments on the pre-trained language models provided by Huggingface (Wolf et al., 2020). For the teacher, we used the 24-layer RoBERTa, and 2 variants of DistilRoBERTa (Sanh et al., 2019) for the student; the original 6-layer DistilRoBERTa, and a 3-layer DistilRoBERTa with half the layers removed. While 24-layer RoBERTa has 355 parameters, this is reduced to 82.1 million in 6-layer DistilRoBERTa, and then to 60.8 million in our 3-layer version. For each model, we appended a linear layer followed by a ReLU activation, a 0.1 dropout layer, and a linear output layer. We stabilized training by following the configurations suggested by Mosbach et al. (2020). That is, we applied the ADAM optimizer (Kingma and Ba, 2014) with a bias correction to avoid vanishing gradients in early training steps. We then trained for 40 epochs with a learning rate of 2×10^{-5} that linearly increases in the first 10% of the total training steps and linearly decays to zero afterward.

4.3 Datasets

In an attempt to evaluate our approach under different settings, we considered datasets of multiple sequence classification tasks. We simulate scarce data settings by artificially creating an initial training set of randomly selected seed examples from the available training data. The language generation model, and the student and teacher classifiers, are then fine-tuned on the sampled training data. The SST-2 (Socher et al., 2013), and Yelp-2 (Zhang et al., 2015) datasets are for binary sentiment classification. TREC-6 (Li and Roth, 2002) is a 6-label question classification dataset, and DBpedia (Zhang et al., 2015) is a 14-label topic classification dataset. Finally, the Microsoft Research Paraphrase Corpus (MRPC), is for a sentence-pair classification task (Dolan and Brockett, 2005), where a model has to predict if the two sentences are semantically equivalent or not. We note that other data augmentation works have fine-tuned GPT-2 on SST-2, Yelp and TREC (Kumar et al., 2020; Anaby-Tavor et al., 2020; Feng et al., 2020). For the SST-2 experiments, from a total of 67,349 samples, we randomly sampled 30 examples per label from the GLUE SST training data (Wang et al., 2018), and evaluated on the provided development set. The full training dataset for Yelp-2 contains 560K samples. Here, we sampled 20 examples per label, making a total of 40 training samples. We then evaluated on the 25,000 test samples. TREC-6 contains 5,452 training examples and 500 testing examples; as this training dataset is not balanced, we randomly sampled less than 1.5 percent of the data for each class, giving us a total of 76 examples. For DBpedia, we sampled just 3 examples per label, from a total of 560K instances, making a seed dataset of 42 training examples. We then evaluated on the 70,000 test samples. Finally, MRPC consists of 3,668 training examples of which we sampled 600 per label, making a total of 1,200 training samples. Our evaluations were on the 1,725 samples test set.

4.4 Configurations

We configured both MCTG and NTTG to 5k iterations, and the top-k sampling to $k = 20$ in all our experiments. We also added a pruning criterion to limit the maximum length of any sequence to

Task	Teacher	Student		Student (Post Distillation)			Student
		layers	Start	MCTG	NTTG	C-BERT	Upper-Bounds
SST-2 # Samples	89.9 (60)	6-layers	78.2 (60)	**86.1** (1508)	85.3 (402)	83.1 (4718)	91.4 (67349)
		3-layers	68.2 (60)	**82.3** (3104)	80.7 (734)	75.5 (4718)	89.4 (67349)
DBpedia-2 # Samples	92.3 (42)	6-layers	80.4 (42)	92 (3976)	**93.1** (1602)	88.8 (5029)	99.3 (560K)
		3-layers	41.1 (42)	**91.6** (3940)	90.6 (3271)	84.1 (5029)	99.2 (560K)
TREC-6 # Samples	89 (76)	6-layers	80 (76)	**88** (2508)	83.4 (537)	80 (4874)	96.8 (5452)
		3-layers	62 (76)	**82** (3133)	81 (594)	78.4 (4874)	95.8 (5452)
MRPC # Samples	84.5 (1200)	6-layers	77.9 (1200)	81.5 (4252)	81.6 (3599)	**82.2** (6200)	85.6 (3668)
		3-layers	69 (1200)	**77.9** (3118)	77.4 (4156)	73.3 (6200)	78 (3668)
YELP-2 # Samples	82.6 (40)	6-layers	**85.2** (40)	80.7 (3124)	81.1 (2074)	81.8 (5040)	95.9 (560K)
		3-layers	73.9 (40)	**78.9** (4278)	77.6 (2828)	76.9 (5040)	94.9 (560K)

Table 1: Teacher-Student Results (in percent, numbers of added examples in parentheses below). The Upper-Bounds are computed after training the student models on the full training datasets without synthetic data.

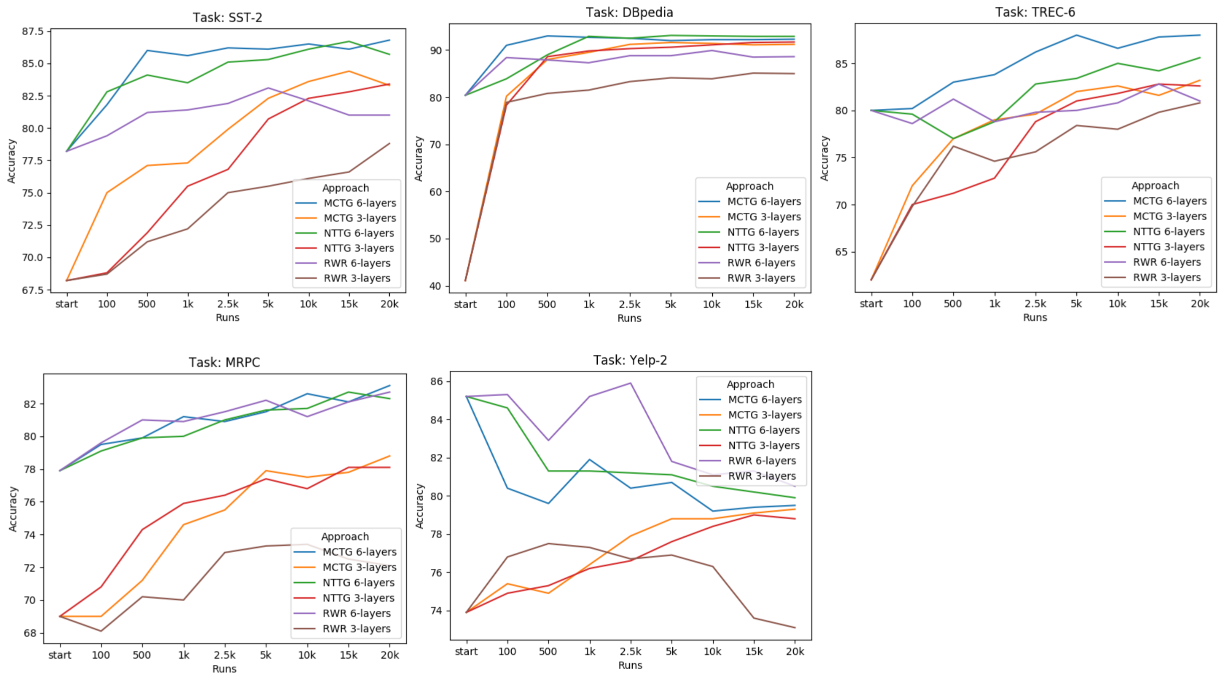

Figure 3: Student's test accuracy after 100, 500, 1k, 2.5k, 5k, 10k, and 20k iterations.

120 tokens. For MCTG, we set the UCB constant from eq. 2 to $C = 3$. As for the reward function, only for the TREC-6 experiments, we added a heuristic (see 'task specific condition' in eq. 6) to condition the first token to be a question word by returning -1 if it was not 'what', 'where', 'when', 'who', 'which', 'why', or 'how'. For each task, we made sure the generated text is of appropriate format for the RoBERTa classification models. This meant, reward = 0 for GPT-2 outputs that were not in the format <|endoftext|>$x_1, \dots x_N$ <|endoftext|> for single input sentence tasks; TREC-6, SST-2, Yelp, and DBpedia. As for MRPC, where the input is 2 sentences, the generated data has to be of format <|endoftext|> $x_1, \dots x_N$, [SEP], $y_1, \dots y_N$ <|endoftext|>. Where $x_1 \dots x_N$ and $y_1 \dots y_N$ are sequences of tokens. After data was generated, we selected the examples with a positive reward, see section 3.3. To avoid high imbalance for the binary datasets, we limited the number of added examples to the size of the minority class. For the multiclass datasets, we limited the selection over the median from the distribution of generated examples per label. We then added the selected data to the initial training data, to form a transfer set. This new transfer set consists of the initial training samples with the generated data pseudo-labeled by the teacher.

4.5 Results

Results for the teacher-student knowledge transfer experiments are in Table 1. We show the test accuracy of both the teacher and student (Pre-Distillation) on the sampled data from section 4.3, and the student after it has been trained on the sampled data combined with the generated data (Post-Distillation). We also compute an upper bound performance by training the 3-layer and 6-layer DistilRoBERTa models on the full training datasets, mentioned in section 4.3. This is to give us an estimate of the performance that can be achieved with as much non-synthetic data as possible. Underneath each accuracy score, in parentheses, is the size of the training data (# Training examples). For the teacher and the student prior to distillation (labeled Start in the table), the data sizes are of the initial training sets, described in section 4.3. As explained in section 4.2, the teacher model is a 24-layer RoBERTa, and the student model, is either a 6-layer DistilRoBERTa, or a 3-layer DistilRoBERTa. The "Start" accuracy is achieved after training only on the initial dataset. For example, the 24-layer RoBERTa trained on the SST-2 dataset, of 60 examples, produced a test score of 89.9. This is a much higher result compared to the accuracies of the 6-layers and 3-layers DistilRoBERTa models that were trained on the same dataset, scoring only 78.2 and 68.2 respectively. We then applied MCTG, from section 3.2, and NTTG, from section 3.4 to generate distillation data. We fixed the total number of iterations to 5k. After each iteration, a sample is generated and only becomes a candidate for distillation if its Δent (equation 5) is positive, and receives a positive reward as per the conditions in equation 6. This means less samples are added to the transfer set from the total generated data.

4.6 Discussion

Overall, results in Table 1 show that our approach works well with either MCTG or NTTG. It is evident that a good teacher can always increase the performance of its student, provided that enough examples achieve a positive Δent (equation 5). This shows that regardless of the generation method, equation 5 remains a key component to our approach. Overall, our approaches, NNTG and MCTG, lead to better performance improvements over the C-BERT baseline. Only in MRPC, the results are similar for the 6-layer student, which could be attributed to a lower performance gap between the student and the teacher. Overall, when compared to the upper-bound results from training the distilled models on the full training datasets, the performances we achieve with distillation are not far off. This might indicate that there is potential room for improvement. With this in mind, the distilled models only utilized a fraction of the full training datasets under each task, as explained in section 4.3. In Figure 3 for each task, we plot the student's performance after iterations 100, 500, 2.5k, 5k, 10k, and 20k. As the number of iterations increase, more examples are generated and thus better performance can be achieved. However, we notice that at a certain point, the increase in performance plateaus. To show the importance of a good teacher, we selected a dataset (Yelp-2), in which the 6-layer student outperforms its teacher. Here, the teacher's overconfidence in incorrect predictions resulted in noisy data, that degraded its student's performance. This shows that the student can only improve as much as its teacher is able to provide good labels. We investigate the stability of the student model, pre- and post-distillation, by

Task	Approach	3-Layers		6-Layers	
		Start	20K	Start	20K
SST-2	MCTG	68.2(±1.67)	83.7(±0.32)	81.85(±1.26)	86.89(±0.396)
	NTTG	67.7(±1)	82(±0.69)	81.3(±0.869)	86.9(±0.4)
TREC-6	MCTG	59.86(±4.97)	83(±0.7)	78.6(±1.77)	88(±0.51)
	NTTG	60.15(±4.68)	82.6(±0.84)	79.08(±1.65)	86.3(±0.8)
MRPC	MCTG	68.5(±2.28)	77.5(±0.55)	79.7(±0.91)	82.7(±0.4)
	NTTG	68.2(±2.69)	77.6(±0.49)	79.5(±0.71)	81.5(±0.7)
DBpedia	MCTG	52.18(±5.06)	91.3(±0.16)	86(±3.7)	92.3(±0.059)
	NTTG	55.7(±4.7)	91.7(±0.13)	85.9(±3.8)	92.8(±0.08)
Yelp-2	MCTG	72.27(±2.84)	79.2(±0.183)	83.29(±1.34)	79.4(±0.175)
	NTTG	72.4(±2.3)	78.5(±0.167)	83.95(±1.07)	79.9(±0.22)

Table 2: Mean(± standard deviation), of test accuracy for 10 student model (3-layers and 6-layers) instances, trained on the initially sampled data and the pseudo-labeled data from the 20k MCTG and NTTG runs.

TREC-6 Examples	**Teacher**	**Student**
What is virtual reality?	DESC	ENTY
What language was originally spoken by the Indians?	ENTY	LOC
Where is your favourite golf course?	LOC	DESC
SST-2 Examples		
a trip from good to bad	NEG	POS
the kind of script worth watching	POS	NEG
a step down from her best years.	NEG	POS

Table 3: Examples of data generated for TREC-6 and SST-2. Wrongly predicted labels are colored in red.

running 10 training instances on the initial data and on the transfer set from the 20k run. In Table 2, we show the mean of the test accuracy of 10 trained instances of the student model. These results are consistent with Table 1. Overall, the augmented data leads to better and more stable models, indicated by the higher accuracy and lower variance. Finally, in Table 3, we show some generated data from the TREC-6 and SST-2 experiments. All four examples are remarkably grammatical, natural, and well-formed.

5 Conclusion

In this paper, we presented an approach for generating text data in order to improve knowledge distillation on small datasets. By selecting examples predicted with the lowest uncertainty by the teacher and the highest uncertainty by the student, we were able to improve the student's performance, sometimes almost to the level of the teacher. Considering the results, we could argue that reward-based language generation can complement or even substitute for heuristic data augmentation approaches in knowledge distillation. We believe that our approach can serve as a baseline for reward-based textual data generation in small data settings. This will hopefully motivate future research to further explore reward-based generation methods.

Acknowledgments

Husam Quteineh acknowledges the support of the Business and Local Government Data Research Centre BLG DRC (ES/S007156/1) funded by the Economic and Social Research Council (ESRC).

References

Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. 2020. Do not have enough data? deep learning to the rescue! In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7383–7390.

Broderick Arneson, Ryan B Hayward, and Philip Henderson. 2010. Monte carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258.

Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256.

C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

Jang Hyun Cho and Bharath Hariharan. 2019. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4794–4802.

Michael Chung, Michael Buro, and Jonathan Schaeffer. 2005. Monte carlo planning in rts games. In *CIG*. Citeseer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

William B Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

Stefan Edelkamp, Max Gath, Christoph Greulich, Malte Humann, Otthein Herzog, and Michael Lawo. 2016. Monte-carlo tree search for logistics. In *Commercial Transport*, pages 427–440. Springer.

Angela Fan, Mike Lewis, and Yann N. Dauphin. 2018. Hierarchical neural story generation. *CoRR*, abs/1805.04833.

Steven Y Feng, Varun Gangal, Dongyeop Kang, Teruko Mitamura, and Eduard Hovy. 2020. Genaug: Data augmentation for finetuning text generators. *arXiv preprint arXiv:2010.01794*.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. 2021. Knowledge distillation: A survey. *International Journal of Computer Vision*, pages 1–31.

Byeongho Heo, Minsik Lee, Sangdoo Yun, and Jin Young Choi. 2019. Knowledge transfer via distillation of activation boundaries formed by hidden neurons. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3779–3787.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*.

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

Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201*.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.

Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. *arXiv preprint arXiv:2003.02245*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations.

Xin Li and Dan Roth. 2002. Learning question classifiers. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Marius Mosbach, Maksym Andriushchenko, and Dietrich Klakow. 2020. On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv:2006.04884*.

Phi-Vu Nguyen, Ali Ghezal, Ya-Chih Hsueh, Thomas Boudier, Samuel Ken-En Gan, and Hwee Kuan Lee. 2016. Optimal processing for gel electrophoresis images: applying monte carlo tree search in gelapp. *Electrophoresis*, 37(15-16):2208–2216.

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

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Husam Quteineh, Spyridon Samothrakis, and Richard Sutcliffe. 2020. Textual data augmentation for efficient active learning on tiny datasets. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. 2014. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*.

Hassan Sajjad, Fahim Dalvi, Nadir Durrani, and Preslav Nakov. 2020. Poor man's bert: Smaller and faster transformer models. *arXiv preprint arXiv:2004.03844*.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

Connor Shorten and Taghi M Khoshgoftaar. 2019. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Raphael Tang, Yao Lu, Linqing Liu, Lili Mou, Olga Vechtomova, and Jimmy Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *arXiv preprint arXiv:1903.12136*.

Antti Tarvainen and Harri Valpola. 2017. Weight-averaged consistency targets improve semi-supervised deep learning results. *CoRR*, abs/1703.01780.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *arXiv preprint arXiv:1905.00537*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

Xing Wu, Shangwen Lv, Liangjun Zang, Jizhong Han, and Songlin Hu. 2019. Conditional bert contextual augmentation. In *International Conference on Computational Science*, pages 84–95. Springer.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*.

Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. 2017. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138.

Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28:649–657.