Multi-task Learning for Natural Language Understanding with Active Task Prioritization using Task-Specific Model Uncertainty

Fei Fang
ffang19@stanford.edu

Ammar Alqatari
ammarq@stanford.edu

Abstract

While multitask learning can improve inference speed as well as performance by learning a shared representation from training multiple related tasks in parallel, it remains challenging to balance the contribution of each task during training for optimal performance across all tasks. In this work, we highlight the discrepancy among convergence rates of different tasks as a less studied and perhaps overlooked aspect of this challenge. We propose *uncertainty-based active task prioritization* as a solution to this problem. Our work builds on previous insight that prioritizing difficult tasks over easier tasks during training leads to improved model performance, as well as the observation that Bayesian model uncertainty is a component of predictive uncertainty that can be reduced with increasing data. Combining these insights, we propose a batch sampling algorithm which actively prioritizes batches from tasks that the network is most uncertain about, by estimating task-specific model uncertainty using Bayesian deep learning methods. More specifically, we define the model's uncertainty about a task as the model's uncertainty about the dev examples of a task. We periodically evaluate the model on dev set during training, estimate uncertainties using Monte Carlo dropout; then update task sampling weights as proportional to an exponential moving average of the raw uncertainty estimates; sample batches from tasks given the updated task sampling weights; and repeat.

We showcase our method in the setting of multi-task natural language understanding (NLU) using three binary text classification tasks and datasets of varied dataset sizes and complexity from the General Language Understanding Evaluation (GLUE) benchmark. We compare our method with two commonly used batch sampling algorithms, and demonstrate that our approach is able to dynamically pace the training progress of different tasks: it not only speeds up the convergence of tasks with high initial uncertainty and slows down the convergence of tasks with low initial uncertainty, but also adjusts the training speed of each task as the relative uncertainties among them change. The dynamic pacing mitigates the discrepancy in convergence rates among tasks as hypothesized. We also show that the improved alignment of convergence rates improve performance on downstream metrics. Overall, across all three tasks, our approach outperforms the other two batch sampling algorithms by at least 0.95-pt in aggregate ROC-AUC, at least 0.87-pt in aggregate accuracy, and at least 0.49-pt in aggregate F1-score.

In summary, our main contributions to multitask learning are as follows:

- A demonstration that the discrepancy among convergence rates of different tasks impact downstream performance and is a challenge in multitask training.
- A principled algorithm for multitask batch sampling that dynamically paces the training progress of each task by actively prioritizing difficult tasks.
- The first application of Bayesian epistemic uncertainty in offline multitask learning to the best of our knowledge.

1 Introduction

Multitask learning is an approach that aims to improve inference speed as well as performance by learning a shared representation from training multiple related tasks in parallel [4, 16]. Multitask learning has been widely explored in many deep learning applications, such as computer vision (e.g. scene understanding) and natural language processing (e.g. text classification), and some multitask models outperform their single-task counterparts [16, 14].

For the scope of this project, we explore multitask learning in the setting of binary classification tasks in natural language understanding (NLU). Multitask learning was first proposed in 1997 and has been studied extensively in computer vision. In recent years, there has been growing excitement about the potential of deep multitask learning in NLU, especially as neural-based NLU is increasingly applied in settings where it's impractical to obtain large amounts of labeled data for a single task. Consider intent classification in a domain-specific conversational AI assistant, for example. Since each application can have its own set of dozens of domain-specific intents, collecting labeled data can be very costly. Furthermore, since inference needs to be done in real-time, it's necessary to have both high speed and strong performance. By leveraging multitask learning, one can combine the classification of all intents into one classifier and thus drastically reduce the computation complexity while boosting the performance via cross-task data sharing.

However, the training of multitask networks is not without its unique challenges. For instance, tasks may compete and negative transfer may occur [26]. In addition, it has been shown that the performance of multitask networks tends to be extremely sensitive to the relative weighting of the tasks' respective losses [16]. A less studied and perhaps overlooked factor is the discrepancy among the convergence rates of different tasks. It is well established that in single-task learning, the same model architecture can converge at different rates when trained on different tasks. The same phenomenon occurs in multitask learning: at a given iteration, the network might have started to overfit for a certain task, while not having converged yet for another. As only one set of model parameters can be selected per training session, this phenomenon makes early stopping and selecting the best parameters difficult, since it would lead to better performance on some tasks but at the cost of others'. Therefore, it is crucial to seek a solution that mitigates the discrepancy among the convergence rates of different tasks.

In this project, we show that one solution is to prioritize the tasks that take longer (i.e., require more data) to converge by giving the network more data from these tasks earlier on during training. To determine a priori which tasks would take longer to converge, we look to the concept of epistemic uncertainty in Bayesian modeling. Epistemic uncertainty, also called model uncertainty, is categorized as "uncertainty in the model parameters that can be explained away given enough data" [16]. In other words, when given more data on a task, the model's uncertainty about said task decreases. We hypothesize that there is a strong correlation between high initial model uncertainty and slow convergence rate: the more uncertain the model is about a given task at the start of training, the more data it needs from said task to converge, and the longer it takes.

Given the interpretation above, we propose a minibatch sampling algorithm which actively prioritizes minibatches from tasks that the network is most uncertain about, by estimating task-specific model uncertainty using Bayesian deep learning methods. We evaluate our method using three binary text classification tasks and datasets from the General Language Understanding Evaluation (GLUE) benchmark [27]:

1. Single-sentence sentiment classification task using Stanford Sentiment Treebank (SST-2) [25]
2. Sentence-pair paraphrase classification task using Microsoft Research Paraphrase Corpus (MRPC) [7]
3. Sentence-pair natural language inference task using Recognizing Textual Entailment (RTE) datasets [5, 2, 12, 3]

Intuitively and statistically (according to SOTA results on these tasks), the tasks increase in difficulty, as the level of semantic understanding demanded of the model increases. We compare our method with two commonly used batch sampling algorithms, and demonstrate that our approach is able to dynamically pace the training progress of different tasks: it not only speeds up the convergence of tasks with high initial uncertainty and slows down the convergence of tasks with low initial uncertainty, but also adjusts the training speed of each task as the relative uncertainties among them

change. The dynamic pacing consequently mitigates the discrepancy in convergence rates among tasks as hypothesized. We also show that the increased alignment of convergence rates improve performance on downstream metrics. Overall, across all three tasks, our approach outperforms the other two batch sampling algorithms by at least 0.95-pt in aggregate ROC-AUC, at least 0.87-pt in aggregate accuracy, and at least 0.49-pt in aggregate F1-score.

In summary, our main contributions to multitask learning are as follows:

- A demonstration that the discrepancy among convergence rates of different tasks impact downstream performance and is a challenge in multitask training.
- A principled algorithm for multitask batch sampling that dynamically paces the training progress of each task by actively prioritizing difficult tasks continually.
- The first application of Bayesian epistemic uncertainty in offline multitask learning to the best of our knowledge.

2 Background and Related Work

Our work draws on techniques from multitask and transfer learning in NLU, Bayesian deep learning, and active learning. We give a brief overview of the definitions and main results we rely on in each of these fields.

2.1 Multitask Learning and Transfer Learning for Natural Language Processing

Transfer learning is the technique of using a model pretrained for a different but related task as the starting point for the task at hand, with the aim to transfer knowledge from the previous task to the new task for more effective learning. This technique can be viewed as sequential multitask learning, and has transformed the field of neural-based NLU, in particular with the advent of Bidirectional Encoder Representations from Transformers (BERT) [6]. BERT is a generic natural language understanding model pretrained on English Wikipedia and BookCorpus [29] in an unsupervised fashion, and can then be fine-tuned for downstream natural language tasks with supervision. At the time of BERT's debut, BERT obtained state-of-the-art results on the GLUE benchmark via single-task finetuning.

Later work has since combined large-scale pretraining and multitask finetuning. One such example is the Multi-Task Deep Neural Network (MT-DNN), which adds an intermediate step of multitask finetuning between pretraining and single-task finetuning for NLU. This approach not only improved the SOTA performance on GLUE, but also resulted in a model that is able to adapt to NLU tasks in other domains with significantly fewer in-domain labels [20]. Our work directly builds upon MT-DNN, and seeks to improve the effectiveness of multitask finetuning. More specifically, in the multitask finetuning step of MT-DNN, the training data is simply prepared by taking the union of datasets from all tasks (which we will refer to as union sampling), despite the fact that the tasks have a wide range of complexity and have datasets of different sizes. In our project, we propose an algorithm that dynamically samples training data in a way that accounts for the varied complexity and data imbalance.

2.2 Bayesian Deep Learning

In Bayesian modeling, a model's predictive uncertainty falls into two general categories: epistemic and aleatoric. Epistemic uncertainty, or model uncertainty, refers to uncertainty about the model parameters which can be reduced by observing more training data. For example, an SVM binary classifier has high uncertainty on data points near its decision boundary, but observing more data can reduce the SVM margin and hence the predictive uncertainty. Aleatoric uncertainty, or data uncertainty, refers to randomness presumed to be intrinsic to the data, and generally remains constant regardless of the size of the training dataset. Aleatoric uncertainty can be further categorized into *heteroscedastic*, or input-dependent uncertainty, and *homoscedastic* uncertainty, which remains constant regardless of input [15]. An example of aleatoric uncertainty is noise in sensor readings, which is irreducible regardless of how much data is observed from the sensor. If sensor readings get noisier as the sensor gets further away from the signal, then distance to the signal is an example of heteroscedastic uncertainty which varies based on input.

Bayesian neural networks (BNN) provide a framework for modeling uncertainty in deep learning. BNNs replace deterministic weights with a prior distribution over model parameters, allowing for uncertainty estimation by marginalizing over an approximate posterior distribution of weights. This can be accomplished using Monte Carlo dropout, where samples of model parameters are collected with multiple forward passes using inference-time dropout [10]. The precision of the uncertainty estimate increases with the number of forward passes. In classification models using softmax outputs, model uncertainty can be estimated by the mutual information between the prediction and the posterior over the model parameters [9].

Bayesian uncertainty modeling has been extensively applied towards improving the robustness of neural network predictions. In addition, it also has applications in improving performance in single-task and multi-task training. In Kendall and Gal [15], it is shown that in single-task deep learning, incorporating aleatoric uncertainty and/or combining it with epistemic uncertainty improves performance on segmentation and depth regression benchmarks in computer vision. The same work further confirms that epistemic uncertainty, estimated using MC Dropout, is gradually reduced throughout training, while aleatoric uncertainty generally remains constant. The application of aleatoric uncertainty has also been explored in the setting of multitask learning. For instance, Kendall, Gal, and Cipolla propose that in multi-task learning, homoscedastic uncertainty can be interpreted as task-dependent uncertainty and thus incorporated into task-specific loss weighting [16]. Yang et. al make use of confidence weighting in online multitask learning and demonstrate improved performance on a number of datasets [28].

To our knowledge, however, epistemic uncertainty has not been used in offline multitask learning. Since epistemic uncertainty is the type of uncertainty which is reduced with more data, we expect that its use for task prioritization can align task convergence rates and improve performance in the multi-task setting, which is the main discussion in our results section (4.4.1).

2.3 Active Learning and Active Task prioritization

Active learning (AL) is a technique used in the data-curation process of machine learning to improve data efficiency. Instead of collecting large amounts of training data up front, which is often costly especially in domains where expertise is required (e.g. medical applications), in AL, the model iteratively trains and queries labels from an expert for the examples that would be most informative from the pool of unlabeled data [17]. The function used to determine each unlabelled example's informativeness is referred to as the acquisition function.

In previous work, the principles of active learning have been borrowed in the setting of multitask reinforcement learning, by sampling the harder tasks more frequently than the simpler ones [24]. The active sampling-based task prioritization scheme is as follows: prior to training, each task is associated with a target performance, which is either gathered from human performance, single-task training, or previous benchmarks. The further the model performance is on a given task from the corresponding target performance, the more likely will the task be learned next. However, a major drawback for this approach is that it can be unrealistic to come up with a reasonable target performance under many "real-life" circumstances, such as in medical applications where acquiring human performance is expensive and data for each task is scarce.

Our work addresses the drawback above by using the concepts of Bayesian AL for a more principled approach. In Bayesian AL, model uncertainty is the measure of informativeness; in other words, the acquisition function maps an unlabelled example to the model's uncertainty about said example. One example of this formulation for classification tasks is BALD (Bayesian Active Learning by Disagreement), where the acquisition function estimates the model uncertainty in terms of mutual information between the model predictions and the model parameters [11]. BALD has since been extended to BatchBALD [17], which is used in our approach, to provide a joint uncertainty estimate over a batch of datapoints instead of a single one. Specifically, we estimate the model's uncertainty of a *task* by estimating the model's uncertainty of the validation set of said task using BatchBALD, and prioritize the tasks that have the highest uncertainty estimates. This approach is theoretically grounded and can be readily applied to many realistic scenarios where there does not exist any previous model or benchmark for the problem at hand.

The idea of prioritizing difficult tasks has also been applied to task loss weighting in multitask learning. In dynamic task prioritization [13], the loss weight of a task is dynamically adjusted throughout

Algorithm 1: Uncertainty-based Active Task Prioritization

$N \leftarrow$ maximum number of iterations
$f \leftarrow$ frequency of dev set evaluation and uncertainty estimation updates
$\{\mathcal{B}_t^{tr}\}_{t=1}^T \leftarrow$ set of minibatches for each task, prepared from $\{\mathcal{D}_t^{tr}\}_{t=1}^T$
$u_{1:T} \leftarrow [1, ..., 1]$ // initialize uncertainty estimates uniformly for each task
for *iter* $\leftarrow 0$ ***to*** N **do**
  if *iter* % $f = 0$ **then**
    if *iter* > 0 **then**
      // update task weights based on uncertainty estimates
      for $t \leftarrow 1$ ***to*** T **do**
        $u_t \leftarrow$ `estimate_model_uncertainty`$(\mathcal{D}_t^{dev})$
      end
    end
    $p_{1:T} \leftarrow$ `normalize_task_weights` $(u_{1:T})$
    // sample next f minibatches
    Sample $\mathcal{T}_f \sim$ Multinomial$(f, p_{1:T})$ // sample f tasks with replacement w/ $p_{1:T}$
    $\mathcal{B}_f \leftarrow []$
    for $t \in \mathcal{T}_f$ **do**
      Sample minibatch $B \sim \mathcal{B}_t^{tr}$ without replacement uniformly
      append B to $\mathcal{B}_f$
    end
  end
  for $B \in \mathcal{B}_f$ **do**
    model.update(B)
  end
end

training and inversely proportional to the model's performance on said task. Performance is evaluated using a unified key performance indicator (KPI) that is applicable to all tasks (e.g., accuracy, F1). Our approach is similar in the way that it can be viewed as selecting model uncertainty as KPI, but at the same time orthogonal as it uses the KPI to adjust the *data* ratio among tasks instead of the *loss* ratio among tasks.

3 Method

We introduce *uncertainty-based active task prioritization*. Our approach shares the hypothesis with [24, 13] that prioritizing difficult tasks is likely to make training more efficient and improve model performance. However, unlike the prioritization schemes in previous work which rely on "hyperparametrized" target performance values, our approach uses model uncertainty computed on each task's validation set as the basis for prioritization.

In all following sections, assume our dataset consists of T tasks $\{\mathcal{T}_t\}_{t=1}^T$ with corresponding training datasets $\{\mathcal{D}_t^{tr}\}_{t=1}^T$ and dev datasets $\{\mathcal{D}_t^{dev}\}_{t=1}^T$. Our general approach for active sampling of batches is described in Algorithm 1. The subroutines `estimate_model_uncertainty` and `normalize_task_weights` in Algorithm 1 could be implemented in a number of ways. For example, the model uncertainty estimate could be obtained using focal loss [23] or spectral-normalized neural Gaussian Processes [18]. We describe our final approach for each in the subsections below.

3.1 Estimating Task Uncertainty

We define the model's uncertainty about a task as the model's uncertainty about the examples in the dev set of said task. A canonical measure of model uncertainty in Bayesian modeling is the mutual information between the prediction and the posterior over the model parameters [9]. Let C be the set of classes in the classifier and $\omega \sim p(\omega|\mathcal{D}^{tr})$ be the model parameters. For a single example x, we compute the model's uncertainty about x using Monte Carlo dropout as follows:

1. Collect K softmax predictions with dropout; this is equivalent to sampling $\hat{\omega}_1, \cdots \hat{\omega}_K \overset{iid}{\sim} p(\omega|\mathcal{D}^{tr})$
2. The posterior prediction is then the mean of all softmax predictions: for all $c \in C$,

$$\hat{p}(y = c|x, \mathcal{D}^{tr}) = \frac{1}{K} \sum_{k=1}^{K} p(y = c|x, \hat{\omega}_k, \mathcal{D}^{tr})$$

3. Approximate the mutual information between the prediction y and the model parameters ω:

$$\begin{aligned}
\mathbb{I}(y, \omega|x, \mathcal{D}^{tr}) &:= \mathbb{H}(y|x, \mathcal{D}^{tr}) - \mathbb{E}_{p(\omega|\mathcal{D}^{tr})}[\mathbb{H}(y|x, \omega, \mathcal{D}^{tr})] \\
&\approx -\sum_{c \in C} \hat{p}(y = c|x, \mathcal{D}^{tr}) \log \hat{p}(y = c|x, \mathcal{D}^{tr}) \\
&\quad + \sum_{k=1}^{K} \sum_{c \in C} p(y = c|x, \hat{\omega}_k, \mathcal{D}^{tr}) \log p(y = c|x, \hat{\omega}_k, \mathcal{D}^{tr})
\end{aligned}$$

In our application, since we would like to estimate the model's uncertainty about a *set* of examples instead of a single example, we replace the estimator in step 3 with BatchBALD, a generalized formulation. BatchBALD computes the mutual information between the joint of a batch of predictions and the model parameters. Formally, given a batch of data points with inputs $\{x_1, ..., x_n\}$, and Bayesian model parameters $\omega \sim p(\omega|\mathcal{D}^{tr})$, the BatchBALD score is

$$\begin{aligned}
\alpha_{\text{BatchBALD}}(x_{1:n}, p(\omega|\mathcal{D}^{tr})) &= \mathbb{I}(y_{1:n}, \omega|x_{1:n}, \mathcal{D}^{tr}) \\
&= \mathbb{H}(y_{1:n}|x_{1:n}, \mathcal{D}^{tr}) - \mathbb{E}_{p(\omega|\mathcal{D}^{tr})}[\mathbb{H}(y_{1:n}|x_{1:n}, \omega, \mathcal{D}^{tr})]
\end{aligned}$$

where y_i is the prediction of x_i for all $i \in \{1, \ldots, n\}$. The estimator of the above scoring function, along with its derivation, can be found in [17]. We base our implementation of the estimator on that found in the Bayesian active learning library (BaaL) [1].

The uncertainty score u_t we assign to a task $\mathcal{T}_t$ is the mean BatchBALD score for all minibatches in D_t^{dev}. Although one would get a more accurate estimate by calculating the BatchBALD score on the joint over the entire dev set, due to memory constraints, that is not tractable and we use the mean across minibatches instead.

3.2 From Uncertainty Estimates to Task Sampling Weights

For each task $\mathcal{T}_t$, given the latest uncertainty estimate u_t, we first compute an intermediate weight

$$v_t^{(\tau)} = \gamma \cdot u_t + (1 - \gamma) \cdot v_t^{(\tau-1)} \tag{1}$$

where τ and $\tau - 1$ correspond to the current and previous eval iteration, respectively, and $\gamma \in [0, 1]$ is the discount factor. The exponential moving average formulation is inspired by momentum, to help smooth out the change in relative task weights across iterations by dampening the oscillations in uncertainty estimates. This also reduces the algorithm's dependence on extremely precise uncertainty estimates. Setting $\gamma = 0$ is equivalent to using raw uncertainty estimates, and as γ increases, the smoothing effect strengthens. We discuss the effects of tuning γ in Section 4.6.

We then normalize the intermediate weights into a probability distribution: for each task $\mathcal{T}_t$,

$$w_t^{(\tau)} = \frac{v_t^{(\tau)}}{\sum_{j=1}^{T} v_j^{(\tau)}}.$$

An alternative approach is to normalize with softmax. However, since it has been shown that softmax tends to skew the distribution towards extremes [10], softmax is not an appropriate choice here, as it would lead to the more uncertain tasks being significantly oversampled at the cost of the less uncertain tasks.

4 Experiments and Results

We evaluate the proposed uncertainty-based active task prioritization algorithm on three tasks from the popular NLU benchmark GLUE: SST-2 [25], MRPC [7], and RTE [5, 2, 12, 3]. We compare our approach with two commonly-used baseline sampling algorithms and demonstrate that our method dynamically paces the training progress of different tasks and thus mitigates the discrepancy in convergence rates among tasks as hypothesized.

Dataset	Task	|Train|	|Downsampled Train|	|Dev|=|Test|
SST-2	single-sentence sentiment	67k	8k	986
MRPC	sentence-pair paraphrase	3.7k	3.7k	204
RTE	NLI	2.5k	2.5k	1.5k

Table 1: Summary of the three tasks: SST-2, MRPC, and RTE.

4.1 Tasks and Datasets

Given the limited resources for this project, instead of evaluating our method on all 9 tasks in GLUE, we limited the scope to a strict subset of tasks, and downsampled training examples when appropriate. Our selection criteria were:

1. The tasks should have different goals;
2. The tasks should have varied amounts of training data;
3. The SOTA results on the tasks should span a wide range.

Criteria 1 and 2 simulate common scenarios where multi-task learning is used. Criterion 3 is in place as a heuristic to make sure that the tasks have a wide range of uncertainties, in order for our Bayesian uncertainty-based methodology to be effectively evaluated. We evaluate each task on three metrics: accuracy, F1 score, and ROC-AUC, with ROC-AUC being the primary metric as it is not sensitive to discrimination threshold tuning and makes for a fairer metric. Since the GLUE benchmark does not use as many metrics and the test labels are held out, we further split the dev set into dev and test evenly. The rest of this section briefly describes the tasks, as summarized in Table 1.

SST-2 The Stanford Sentiment Treebank [25] consists of sentences from movie reviews with human-annotated binary labels that indicate whether the review is positive or negative [27]. This is a **single-sentence classification** task: given a sentence, the model predicts whether it expresses positive or negative sentiment.

MRPC The Microsoft Research Paraphrase Corpus [7] consists of sentence pairs automatically extracted from online news articles, with human-annotated binary labels that indicate whether the two sentences are semantically equivalent [27]. This is a **sentence-pair classification** task: given two sentences, the model predicts whether they are paraphrases of each other.

RTE The Recognizing Textual Entailment dataset [5, 2, 12, 3] combines a series of textual entailment challenges and consists of sentence pairs from news and Wikipedia [27]. This is a **sentence-pair classification** task: given a text and a hypothesis, the model predicts whether the hypothesis is logically entailed by the text.

4.2 Baseline Algorithms

This section describes the two baseline sampling algorithms against which we compare our approach. Assume our dataset consists of T tasks $\{\mathcal{T}_t\}_{t=1}^T$ with corresponding training datasets $\{\mathcal{D}_t^{tr}\}_{t=1}^T$.

Uniform Sampling At each iteration t,

1. Sample a task $\mathcal{T}_j$ by sampling $j \sim \{1, \cdots T\}$ uniformly;
2. Sample a minibatch $\mathcal{B}$ from $\mathcal{D}_j^{tr}$;
3. Update model parameters based on $\mathcal{B}$, using the loss function associated with $\mathcal{T}_j$.

This is the approach used in the training procedure of the pre-BERT predecessor of MT-DNN [19].

Union Sampling For each epoch, form a single training set $\mathcal{D}^{tr} := \bigcup_{i=1}^T \mathcal{D}_i^{tr}$ by merging the training set for each task. After shuffling $\mathcal{D}^{tr}$, construct minibatches as in single-task learning. This is the approach used in the training procedure of MT-DNN [20].

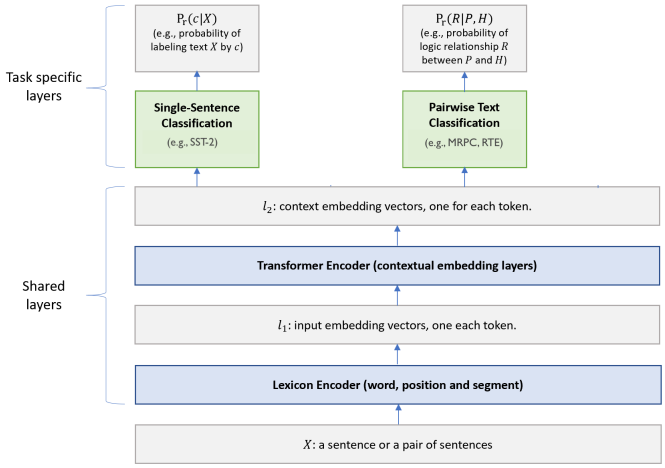

Figure 1: Architecture of the MT-DNN model. The lower layers (including all BERT layers) are shared across all tasks, and the top layers are task-specific output layers. [20]

4.3 Model Architecture and Implementation Details

For all algorithms, we performed our experiments using the Multi-Task Deep Neural Network (MT-DNN) architecture, as shown in Figure 1 [20]. We used cross-entropy loss as the objective function for all tasks. We used the pretrained $\text{BERT}_{\text{BASE}}$ to initialize the shared Transformer Encoder of MT-DNN, then performed multitask finetuning on all tasks. We used the same hyperparameters as outlined in the Implementation Details in Liu et al. [20], but did not perform single-task finetuning following multitask finetuning.

We evaluated on dev sets and updated uncertainty estimates every 200 iterations. For uncertainty estimation, we collected $K = 8$ stochastic samples using Monte Carlo dropout for each example in the dev sets. For the discount factor γ used in normalizing uncertainty estimates into a probability distribution, we performed a hyperparameter sweep across [0, 0.125, 0.25, 0.375, 0.5]. Uncertainties were estimated for all three algorithms for the purpose of analysis, but only used in our uncertainty-based active task prioritization. We did three runs per algorithm (and per associated choice of γ in the case of uncertainty-based active task prioritization).

4.4 Main Results

We compare uncertainty-based active task prioritization to the two baseline sampling algorithms described in Section 4.2 in terms of convergence rates of different tasks as well as performance on downstream metrics. For uncertainty-based active task prioritization, we selected the discount factor $\gamma = 0.5$ based on lowest aggregate dev loss across three runs. The results are shown in Figure 2 and Table 2.

4.4.1 Convergence rates of different tasks and their alignment

The plots in Figure 2 show that our approach, *uncertainty-based active task priorization*, results in *dynamic pacing*: it speeds up the convergence of tasks with high initial uncertainty (i.e., RTE, MRPC) and slows down the convergence of tasks with low initial uncertainty (i.e., SST-2). For instance, the dev loss of RTE when using active task priorization achieves its minimum almost twice as fast as when using union sampling, as does the dev loss of MRPC. More importantly, active task prioritization is able to dynamically adjust the training speed as the relative uncertainties among tasks change. Compare the sampling weights of MRPC and SST-2 with active task prioritization over time for example. The weight of MRPC is consistently higher than that of SST-2 from the beginning to 1000 iterations in, when the dev loss of MRPC reaches its mininum, while SST-2 still has a little room for improvement. Immediately afterwards, the weight of SST-2 surpasses that of MRPC, and remains so until the end of training. Note that the relative ordering of weights changes despite the fact that the dev loss of MRPC remains notably higher than that of SST-2, and that unlike previous work,

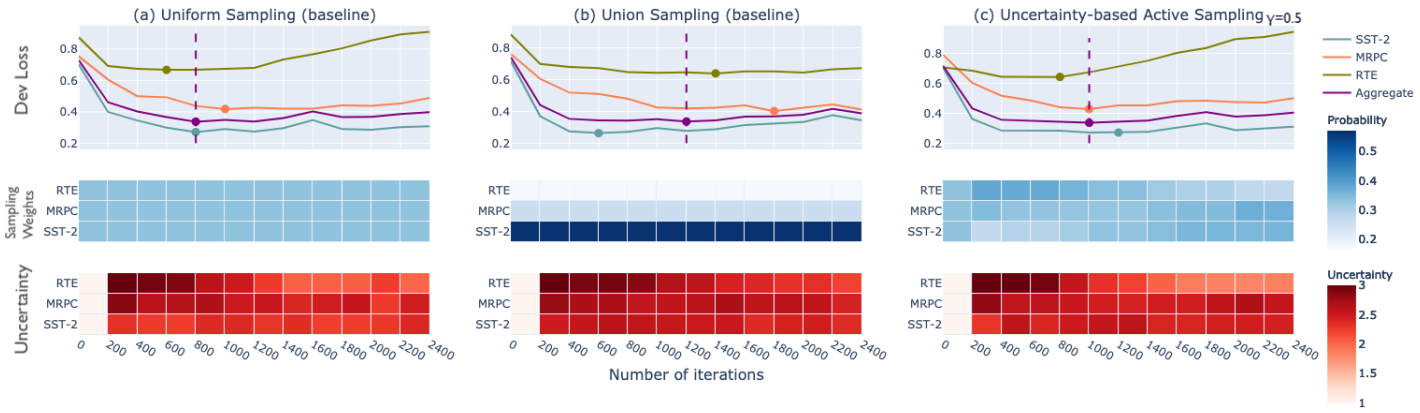

Figure 2: **Comparison of the convergence behavior of various batch sampling algorithms.** The x-axis denotes the number of training iterations. (Top; line plot) Per-task loss and Aggregate (i.e., unweighted per-sample) loss on the dev set during training. Lower is better. The marker on each loss curve corresponds to the minimum. A vertical line is drawn through the number of iterations with the lowest aggregate loss to facilitate comparison of convergence rates. (Middle; tiles) Sampling weight of each task during training. Darker colors denote higher probability and thus higher priority. (Bottom; tiles) Uncertainty estimate of each task during training. Darker colors denote higher uncertainty.

the weighting algorithm has no knowledge of any externally-set target or chosen KPI to optimize for. This corroborates our hypothesis that model uncertainty alone can be sufficient as signal to pace the model's learning.

The dynamic pacing in turn results in closer alignment of the convergence rates of various tasks. This is markedly clear when comparing our approach (Fig. 2c) against union sampling (Fig. 2b). When compared against uniform sampling, it may seem at first glance that the convergence rates are aligned to the same extent, where the dev losses achieve minima no more than 2 evaluation steps apart. However, upon closer inspection, we shall highlight that in the case of active task prioritization, SST-2 has in essence converged by the time the aggregate dev loss reaches its minimum: the dev loss of SST-2 (teal) at the point where aggregate dev loss is lowest is only higher than the lowest dev loss of SST-2 by 0.0002 (0.2715 at 1000 steps vs. 0.2713 at 1200 steps).

4.4.2 Performance on downstream metrics

The results in Table 2 show that uncertainty-based active task prioritization outperforms the other two algorithms on the test set on all three metrics: ROC-AUC, accuracy, and F1 score. On a task-by-task basis, relative to the two baselines, it outperforms on RTE across all metrics; underperforms on SST-2 across all metrics; and performs in between the two baselines on MRPC, ranking first in terms of ROC-AUC and second to uniform sampling in terms of the other metrics.

Its lead in RTE and lack thereof in SST-2 are both unsurprising, as uncertainty-based active task prioritization places the most emphasis on RTE and the least emphasis on SST-2 in the beginning of training. More concretely, the task sampling probability of SST-2 in active task prioritization starts out at 0.27, compared to 0.33 with uniform sampling and 0.57 with union sampling. Meanwhile, the task sampling probability of RTE in active task prioritization starts off at 0.38, compared to 0.33 with uniform sampling and 0.18 with union sampling. Although not unexpected, this does suggest that the active task prioritization might over-prioritize difficult tasks to an extent that the easier tasks get overlooked. This is only a tentative hypothesis since the performance on SST-2 is rather saturated, and would need to be verified on a different set of tasks in future work.

		SST			RTE			MRPC			Aggregate (mean)		
	Method	AUC	Acc.	F1	AUC	Acc.	F1	AUC	Acc.	F1	AUC	Acc.	F1
Dev	Uniform sampling	96.24	90.37	90.84	72.46	67.15	71.06	88.41	**83.17**	**88.61**	85.70	**80.23**	**83.51**
	Union sampling	**96.37**	**90.97**	**91.34**	72.02	66.18	68.61	88.37	**83.17**	88.21	85.59	80.11	82.72
	Active sampling (ours, $\gamma = 0.5$)	96.32	90.37	90.88	**73.89**	**69.08**	**71.91**	**88.71**	80.72	86.86	**86.31**	80.06	83.22
Test	Uniform sampling	96.07	90.03	91.23	69.96	65.47	69.06	85.02	**79.74**	**86.28**	83.68	78.41	82.19
	Union sampling	**96.72**	**91.20**	**92.18**	69.35	62.11	62.97	85.96	78.76	85.12	84.01	77.36	80.09
	Active sampling (ours, $\gamma = 0.5$)	96.14	90.33	91.44	**71.89**	**68.11**	**70.76**	**86.85**	79.41	85.85	**84.96**	**79.28**	**82.68**

Table 2: **Comparison of the performance of various batch sampling algorithms on downstream metrics**. Each algorithm is evaluated on the dev and test sets for all tasks using the model checkpoint that achieves the highest aggregate dev ROC-AUC. The best result on each task is in **bold**. The results are averaged across three runs.

4.5 Trends in dev uncertainty during training

From Figure 2, we observe the following trends in dev uncertainty for each task during training:

1. RTE: The model uncertainty significantly reduces with increasing exposure to training data. This trend is present if we look at the uncertainty with each sampling algorithm in isolation, and even more apparent if we contrast the uncertainty with union sampling, where RTE has the lowest sampling weight throughout, and that with uncertainty-based active sampling, where RTE has the highest sampling weight in the first half of training.
2. MRPC: The model uncertainty steadily reduces with increasing exposure to data in the first half for all three algorithms, and plateaus in the second half for all but uniform sampling.
3. SST-2: The model uncertainty remains generally constant throughout training.

Although trends 1 and 2 are consistent with previous work on Bayesian deep learning that model uncertainty decreases with increasing data, trend 3 seems to contradict that. However, we note the important fact that the model at iteration 0 has been pretrained on large amounts of English data and has already learnt a representation for generic English [22]. Since SST-2 consists of movie reviews, which are usually not written in domain-specific language, and the task is simply to classify semantics into two polarities, the knowledge from pretraining transfers extremely fast. In fact, it has been shown that a model initialized with BERT-base finetuned with merely 800 labeled examples from the IMDB sentiment classification dataset [21] is able to achieve a 0.946 ROC-AUC score [8]. With this in mind, the initial uncertainty of each task can also be interpreted as a measure of distance from the task to the pretraining task. Therefore, trend 3 does not contradict our hypothesis but support it: in the pretraining process of BERT, the model's task of masked language modeling is latently very similar to the sentiment classification task of SST-2, and in some sense has "seen data" (i.e., gained knowledge) about SST-2, which accounts for the low and constant model uncertainty about SST-2 throughout training.

4.6 Effects of tuning the discount factor γ

The discount factor γ used in transforming uncertainties into weights is the only hyperparameter in our algorithm. The larger γ is, the smoother the sampling weights are, as shown in row 2 of Figure 3. The effect is most noticeable for the sampling weight of MRPC, and as γ goes from 0 to 0.25, the model converges faster on MRPC. As γ goes from 0.25 to 0.5, although the model does not converge faster on MRPC, it does converge faster on SST-2. We attribute this to the cross-data sharing and inductive transfer between related tasks in multitask learning. Recall that MRPC is a semantic equivalence classification task and that SST-2 is a sentiment classification task. Both can be viewed as a semantic clustering problem in a latent vector space, with the former being more complex than the latter as the former requires more granular clustering. As a result, it is reasonable that as the model learns more about the more complex problem due to increased exposure to training data from MRPC, it gets better at the less complex problem, i.e., SST-2, without having to directly see large amounts of data from SST-2.

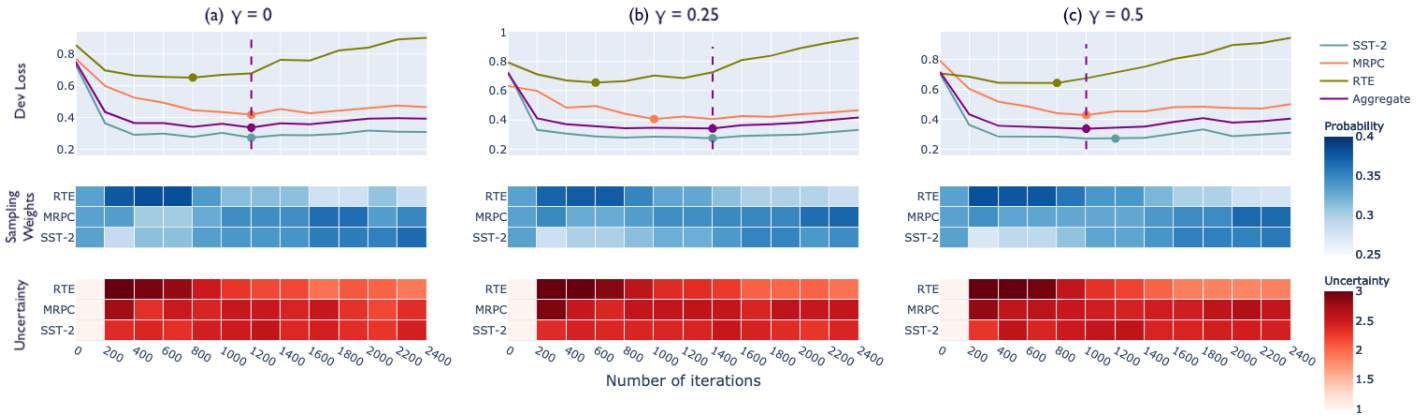

Figure 3: **Comparison of the training behavior of uncertainty-based active task prioritization with different choices of discount factor γ.** Legends and corresponding meanings are the same as those in Figure 2.

The optimal value of γ for the set of tasks at hand is close to 0.5. Recall from Eq. 1 that if γ is too large, then the weight puts too much focus on past uncertainty estimates. As a result, the sampling weights become too biased by initial uncertainty estimates and would not be able to as nimbly adjust the relative training speeds of tasks even if the relative uncertainties among tasks change. This would take away the dynamic pacing property described in 4.4.1, which is undesirable. For the tasks we have chosen, $\gamma = 0.5$ is able to smooth out the uncertainty estimates without placing excessive emphasis on initial uncertainty estimates.

5 Conclusion

In this work, we proposed uncertainty-based active task prioritization as a method for mitigating discrepancies in convergence rates for different tasks in multitask learning. We showed that actively prioritizing batches from tasks with higher uncertainty leads to dynamic pacing in training, where tasks with high initial uncertainty converge faster and tasks with low initial uncertainty converge more slowly compared to naively merging datasets from all tasks into one or uniformly sampling batches from tasks. Although our analysis shows a trade-off on task-specific performance, where active task prioritization can over-prioritize difficult tasks at the expense of easier tasks, our method improves performance on aggregate metrics. Our results highlight task convergence discrepancies as an overlooked factor in multitask training, and demonstrate a novel application of Bayesian uncertainty modeling in multitask learning. For future work, we hope to evaluate this method more thoroughly using different model architectures (e.g., without transfer learning) and benchmarking against other task prioritization techniques that have been proposed. We also hope to iterate on the uncertainty weighting to prevent overprioritization of difficult tasks, by experimenting with integrating epistemic uncertainty into loss weighting as well. Finally, we hope to generalize our method beyond the scope of classification tasks to multitask settings where there exist both regression and classification tasks.

6 Source code

`https://github.com/feifang24/mtl-epistemic-uncertainty`

References

[1] Parmida Atighehchian, Frederic Branchaud-Charron, Jan Freyberg, Rafael Pardinas, and Lorne Schell. Baal, a bayesian active learning library. `https://github.com/ElementAI/baal/`, 2019.

[2] Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. The second pascal recognising textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 6, pages 6–4. Venice, 2006.

[3] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. In *TAC*, 2009.

[4] Rich Caruana. Multitask learning. *Machine Learning*, 28, 07 1997.

[5] Ido Dagan, Oren Glickman, and Bernardo Magnini. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer, 2005.

[6] 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.

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

[8] Fei Fang and Zihan Xie. Weak supervision in the age of transfer learning fornatural language processing. 2019.

[9] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning, 2016.

[11] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. *CoRR*, abs/1703.02910, 2017.

[12] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics, 2007.

[13] Michelle Guo, Albert Haque, De-An Huang, Serena Yeung, and Li Fei-Fei. Dynamic task prioritization for multitask learning. In *European Conference on Computer Vision*, pages 282–299. Springer, 2018.

[14] Jeremy Howard and Sebastian Ruder. Fine-tuned language models for text classification. *CoRR*, abs/1801.06146, 2018.

[15] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *CoRR*, abs/1703.04977, 2017.

[16] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, 2018.

[17] Andreas Kirsch, Joost van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *CoRR*, abs/1906.08158, 2019.

[18] Balaji Lakshminarayanan, Dustin Tran, Jeremiah Liu, Shreyas Padhy, Tania Bedrax-Weiss, and Zi Lin. Simple and principled uncertainty estimation with deterministic deep learning via distance awareness. 2020.

[19] Xiaodong Liu, Jianfeng Gao, Xiaodong He, Li Deng, Kevin Duh, and Ye-yi Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 912–921, Denver, Colorado, May–June 2015. Association for Computational Linguistics.

[20] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *CoRR*, abs/1901.11504, 2019.

[21] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[22] Christopher D. Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 2020.

[23] Jishnu Mukhoti, Viveka Kulharia, Amartya Sanyal, Stuart Golodetz, Philip H. S. Torr, and Puneet K. Dokania. Calibrating deep neural networks using focal loss, 2020.

[24] Sahil Sharma, Ashutosh Jha, Parikshit Hegde, and Balaraman Ravindran. Learning to multi-task by active sampling, 2017.

[25] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. 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, 2013.

[26] Trevor Standley, Amir Roshan Zamir, Dawn Chen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? *CoRR*, abs/1905.07553, 2019.

[27] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. 2019. In the Proceedings of ICLR.

[28] Peng Yang, Peilin Zhao, Jiayu Zhou, and Xin Gao. Confidence weighted multitask learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5636–5643, Jul. 2019.

[29] Yukun Zhu, Ryan Kiros, Richard Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.