When is Deep Multitask Learning Effective in NLP? An Exploration on Cyberhate in Wikipedia

Abraham Starosta
Stanford University
starosta@stanford.edu

Abstract

Although Multitask Learning (MTL) has been applied successfullly to a wide range of tasks, **when** to expect MTL gains in NLP largely remains an open question. We explore this question through a study of cyberhate on Wikipedia, with a dataset that poses common challenges in real-word applications: noisy labels, class imbalance, and varying dataset sizes across tasks. Our main contributions are: (1) providing evidence that when successful, MTL benefits from large auxiliary datasets tightly related to the main task. We see a gain of 0.0065 in ROC AUC on the main task as we increase the size of the auxiliary task. If that was the average gain for all tasks, it would take a Kaggle competitor from the 2000th place right to the 1st place on the public Kaggle leaderboard. (2) We observe that class reweighting or rebalancing are not the answer when you try to mitigate against a highly imbalanced dataset. (3) Our results corroborate the regularization effects of MTL.

1 Introduction to MTL in general

Although MTL has received lots of attention in the past few years, people are often surprised to learn that it has been around for a couple of decades. [Ratner. A(2018)] explains the main reasons why MTL has received an increasing amount of attention in the era of deep learning:

- *MTL amortizes the cost of data acquisition*: organizations can pool labels across multiple tasks, potentially improving the performance of all tasks.
- *Most real-world problems involve solving related subtasks*: for example, many text classification tasks involve a hierarchical ontology with different levels of granularity.

[Caruana(1997)] defines MTL as a way to improve generalization by using domain information in related tasks; what is learned for each task can help others perform better. It does this by co-learning N tasks in parallel, using a shared representation. Although MTL can focus on improving the performance of all tasks at once, the classic approach is to have a "main" task which we are focusing on optimizing and a set of "auxiliary" tasks which we hope boost performance for the main task. It is likely that there is an auxiliary task that can help your main task perform better.

It is worth briefly mentioning the relationship between MTL and Transfer Learning. Transfer learning is somewhat different because given N - 1 "source" tasks, its only goal is to transfer this knowledge in order to perform well on the N^{th} "target" task whereas in MTL, all tasks are trained together. Ultimately, both boil down to solving a multi-objective optimization problem. [Mou. L(2016)] found that MTL is very similar to transfer learning in terms of model performance.

[Ruder(2017)] explains that there are two main methods for implementing deep MTL: hard or soft parameter sharing.

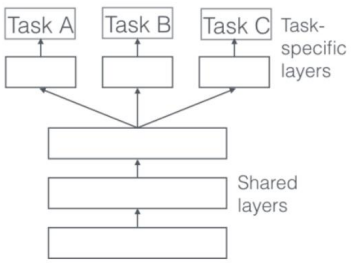

Figure 1: Hard parameter sharing: all tasks share the same hidden layers. Picture from [Ruder(2017)]

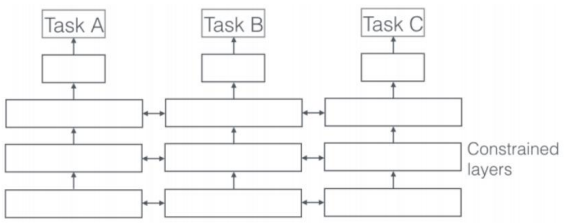

Figure 2: Soft parameter sharing: each task has its own parameters and regularization is used to encourage parameters to be similar across tasks. Picture from [Ruder(2017)]

2 Related work: MTL in NLP

Studies have mixed results on when MTL is expected to be successful. This might be explained by [Mou. L(2016)], finding that transferability in NLP "is different from other domains like image processing" so our conclusions from other subfields in AI do not necessarily hold in NLP.

Some studies focus on task dataset sizes. [Luong. M(2016)] studies MTL within a Seq2Seq architecture applied to Machine Translation. They trained a large Machine Translation main task with small, medium, and large auxiliary tasks. MTL generated the largest gains for the main MT task when trained with a small auxiliary task. However, it is unclear how closely related the large auxiliary task is to the main task used. In contrast, [Benton. A(2017)] shows bigger MTL gains for settings with limited main task data.

Other studies focus on training properties and label distributions. [Bingel. J(2017)] finds that MTL is more successful when the loss curve for the main task plateaus quickly. [Martinez. H(2017)] focuses more on the label distributions and finds that "when successful, auxiliary tasks with compact and uniform label distributions are preferable."

The closest study to our approach is [Kerinec E.(2018)]. The researchers published experiments with MTL on the 20 Newsgroup benchmark dataset, in which documents are classified into 20 loosely related classes. The classes can be represented as a hierarchy where the top classes *(recreation, computer, science, talk, miscellaneous, etc)*, each with a varying number of sub-classes. The authors then choose different task pairs and correlate different features to the amount of MTL gains compared to a baseline single task model (STL). They looked at the following features: Jensen-Shannon Divergence (JSD), gradients of the loss curve at different training epochs, logarithmic fits to the training loss curves, and type-token ratios. They found that the JSD between tasks is the most informative feature, with the loss curve gradients being the second most informative. However, we do have reservations on this study's conclusion on the importance of JSD as a predictor for MTL gains. We have observed that JSD is highly dependent on the size of the task datasets. When one corpus is very small, the JSD tends to be larger because there is just not enough text in the small corpus to be representative of general language properties. In the 20 newsgroup dataset, where task sizes vary substantially, the JSD could be too noisy.

3 Data

3.1 Why this dataset?

We chose to work with **Kaggle's Wikipedia Toxic Comment dataset** because with its noisy task hierarchy and class imbalance, it resembles what an engineer would encounter in the real world. One of our goals is to gather insights that would translate to real-world settings.

3.2 Dataset Overview

The dataset provides a total of 153,165 training samples. Each sample is labeled as *Toxic, Severe Toxic, Obscene, Insult, Threat, Identity Hate.*

Because labels were crowdsourced, the dataset has what we would call a **noisy** task hierarchy.

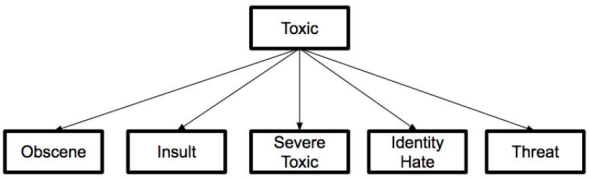

Figure 3: Task Hierarchy.

The vast majority of the labels follow this hierarchy, although a small percentage of 5% do not. For example, most samples labeled as *Obscene* are also labeled as *Toxic*, but a minority might be labeled as *Obscene* but not *Toxic*.

In the table below we provide the class distribution:

Task	Positive Samples	Percentage
Toxic	15,294	9.9%
Obscene	8,449	5.5%
Insult	7,877	5.1%
Severe Toxic	1,595	1.0%
Identity Hate	1,405	0.9%
Threat	478	0.3%
Total	153,165	100.0%

It is evident that the positive labels are very sparse, and this is one of the challenges we attempt to tackle.
Now, to give you a sense of the contents in the comments, since they are quite interesting, we would like to share word clouds for the largest classes.

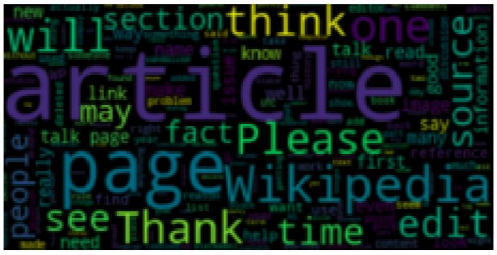

Figure 4: Normal.

Figure 5: Toxic.

Figure 6: Identity Hate.

We can see in the image above the words commonly used in toxic comments. This is a large problem, not only in Wikipedia, but also in other platforms like Twitter, Youtube, and Facebook. Hatespeech is a problem worth working on.

4 Data Processing

4.1 Text Preprocessing

1. Comments are first tokenized using *nltk*'s word tokenizer.

2. Then they are stemmed with *nltk*'s Porter-Stemmer.

3. Tokens longer than 30 characters are filtered out.

4.2 Text Featurization

We use *sklearn*'s TfidfVectorizer on unigrams, filtering out english stopwords and keeping only the top 10,000 features.

There are certainly other featurization strategies such as using word embeddings that would likely improve the model performance, but the goal of our experiments is to study the behavior of MTL models. Therefore, we have attempted to use the most standard featurization strategy possible.

5 Models

5.1 MTL Model Architecture

- Hard parameter sharing: tasks share middle representations.

- Input layer takes in 10,000 dimensional samples.

- 2 hidden layers with 10 units each. We avoid making the network too to avoid overfitting.

- Number of output heads is equal to the number of tasks being learned.

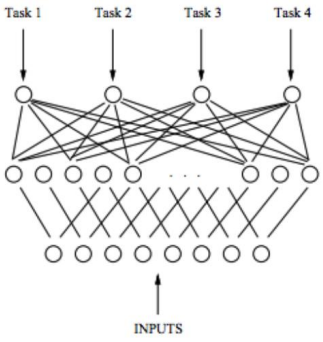

Figure 7: Our MTL architechture. Picture from [Caruana(1997)]

5.2 STL Model Architecture

- Input layer takes in 10,000 dimensional samples.
- 1 hidden layer with 5 units.

5.3 Code Framework

We use Snorkel MeTaL's easy-to-use interface to train and evaluate MTL models as well as STL models [Ratner. A(2019)]. The neural networks are built using PyTorch. This is a project built and maintained by Stanford's Dawn Lab. We made our best to contribute to the MeTal codebase and successfully added a few features as well as proposed multiple enhancements.

5.4 Training

We split the data with a 80/20 scheme. Kaggle tests our predictions against their test set so we use that as our test set.

Both our STL and MTL models are trained with the same hyperparameters:

- Learning rate: 0.0001
- 20 epochs
- Batch size of 32
- No dropout or ℓ_2 regularization.

Our MTL model treats the main and auxiliary tasks equally. Once the batch size is set, a batch of that size for each task is calculated, contributing some amount to the loss, and then a train step is taken. After each epoch, we compare the MTL model's validation score for the main task against previous epochs, and save the one with the best score. Thus, we optimize for the best MTL model for the main task. This is the same for the baseline STL model training process, where we keep the model with the best validation score for the single task.

6 Experiments and Discussion

6.1 Jensen-Shannon Divergence

Since multiple papers suggest that JSD between tasks has high correlation to MTL gains, we wanted to get a sense of the JSD between our tasks.

To compute the JSD, we created a probability distribution over all unigrams in the train sets of each task. Then, we normalize them with a softmax. The JSD values are very small (below 10^3) so we normalize those values by the maximum JSD value in order to facilitate a visual inspection.

Figure 8: JSD between task training sets (using unigram count distribution).

Since all the comments come from the parent *Toxic* class, we would expect JSD to be small and similar for all tasks pairs. However, we observe that the three largest tasks (*Toxic*, *Obscene*, *Insult*) have much lower JSD compared to other task pairs.

In addition, we see in general that the smaller tasks have higher JSD. Except for the *Toxic/Insult* task pair, JSD increases as the dataset size decreases. We believe this casts doubt on the utility of JSD as a predictor for MTL gains because it seems to be tightly correlated to dataset size.

6.2 STL Baselines

We trained STL models on each task in order to have a baseline performance to which we can compare MTL performance.

Task	STL F1 Score
Toxic	0.75
Obscene	0.78
Insult	0.65
Severe Toxic	0.43
Identity Hate	0.42
Threat	0.37

With our simple scheme we got an ROC-AUC of 0.9692 on the test set. For context, this is roughly in the top 75% of Kaggle, which means our data split, basic featurization strategy and STL models are a good place to start. Since our goal was not to optimize overall score, but specifically study gains from MTL, we left it at this.

6.3 MTL Gains

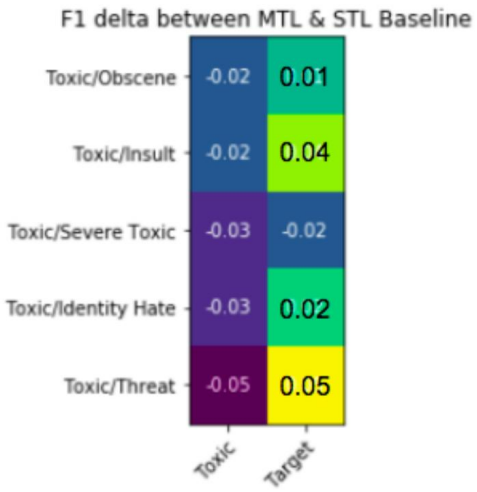

Figure 9: F1 MTL gains for task pairs with Toxic as auxiliary task.

Insights:

- *Auxiliary task is always hurt*: we believe this is partly because of our training process, in which we save the model that performs best on the main task. This opens another line of inquiry on how to mitigate this in the case where you care about improving performance for all tasks instead of focusing on a main task.
- *All main tasks have gains except for Severe Toxic:* this is actually very interesting because you would expect *Severe Toxic* to benefit from the *Toxic* auxiliary task, since you would expect the features to be tightly correlated. This is hard to explain, but our hypothesis is that the features are so correlated that this causes overfitting.
- *Insult received a relatively large gain:* this could have been expected since we previously found that based on JSD, *Insult* was the most closely related task to our auxiliary *Toxic* task. Although JSD is tightly correlated to dataset size, this does provide evidence in favor of JSD's importance in predicting MTL gains.

6.4 MTL gains with smoothly growing auxiliary task

In the previous experiment, we saw that *Insult* was relatively successful for MTL. Thus, we will keep studying the effect of the number of auxiliary positive examples using the *Insult* task.

In this experiment, we start with a roughly equal number of positive examples for both the main and auxiliary task, and then gradually increment the number of auxiliary positive examples.

While using the same validation set and training parameters constant, we smoothly increase the number of auxiliary positive examples and track its effect on the main task performance.

We can see the **main task gains increase** as we include more auxiliary positive examples.

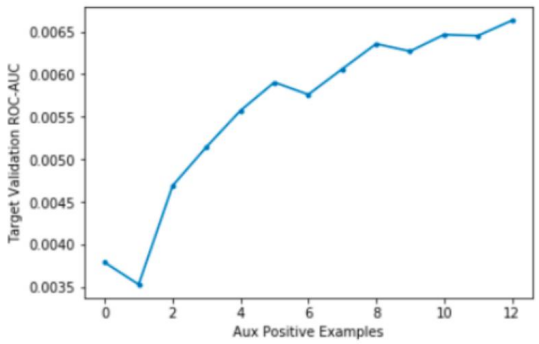

Figure 10: The gains on the main task increase as the number of auxiliary positive examples (x1000) increases.

For context, if the average task performance improved by that amount, it would be the differ-

ence between the 1st place and the 2000th place on the public Kaggle leaderboard.

Our initial hypothesis was that MTL gains would increase as the amount of auxiliary samples increase. The reason lies in the close relationship between MTL and Transfer Learning. It is widely accepted that Transfer Learning is effective when there is a vast amount of data for the auxiliary task, but limited data for the main task. Since MTL and Transfer Learning are on the same spectrum, in which multiple losses are being optimized, it was likely that this property would hold for both.

Another interesting observation is that **train loss increases** while the **validation loss decreases**.

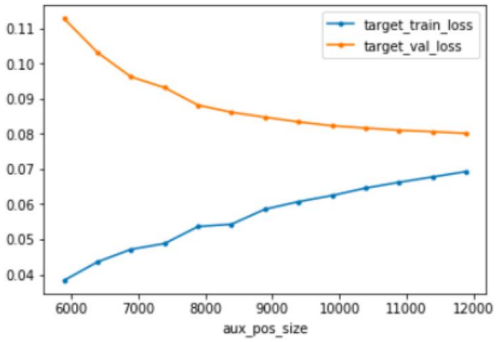

Figure 11: Orange: Validation loss. Blue: Train loss.

This shows MTL's effectiveness as a regularizer. MTL introduces inductive bias by preferring a model that explains multiple tasks, which reduces the chance of overfitting.

6.5 MTL: addressing class imbalance by reweighting and oversampling

Our dataset labels are highly imbalanced towards the negative class, so we attempted to fix that imbalance on the MTL setting with two strategies: class reweighting, and oversampling of positive samples.

The following table shows the F1 score delta of MTL over STL for reweighting and oversampling.

Task	Reweighting	Oversampling
Obscene	0.03	0.01
Insult	-0.17	-0.15
Severe Toxic	0.01	-0.04
Identity Hate	0.02	-0.01
Threat	0.08	0.01

We saw some tasks improve and some worsen, but overall, it seems that neither technique consistently improved results over our STL baselines. Reweighting seems to help in more cases than oversampling, but the gains don't seem to be conclusive. We believe class imbalance in MTL should be tested in multiple datasets to get a better conclusion.

There might be multiple reasons for this. Reweighting causes overfitting in some cases like for the *Insult*, and oversampling when the class imbalance is so large seems to only add redundant information.

7 Conclusions

We have investigated MTL empirically on an NLP application with a noisy task hierarchy, with highly related tasks and imbalanced labels. We have gathered these insights:

- The more samples for the auxiliary task, the higher the gains from MTL on the main task.
- We have corroborated that MTL serves as an effective regularizer.
- Class reweighting or positive oversampling did not help MTL with the problem of class imbalance.
- JSD seems to correlate highly with corpus sizes when the sizes are small and have high variance. An exploration into this issue would be worth it for the sake of research reliability.

Given that our experiments were done only on one dataset, these are certainly not final conclusions.

In terms of insights for fighting hate speech, we have observed gains from MTL and thus we propose leveraging MTL to fight hatespeech.

8 Future Work

Firstly, we plan to run the same experiment, but run it on all main/auxiliary task pair combinations.

Then, we plan to run our experiments on other text datasets with similar characteristics in order to increase the confidence primarily in our conclusion that MTL benefits from large auxiliary datasets.

Afterwards, these are the following questions that we plan on exploring:

- What is the main factor in predicting gains from MTL?
- How do you ensure all tasks get a boost from MTL?
- How useful is JSD as a measure of task similarity in an NLP context?
- How does class imbalance affect MTL and how do you mitigate it?

9 Acknowledgements

I would like to thank Alex Ratner for offering his invaluable feedback at a weekly manner, and Stanford's InfoLab for offering constant support. I am lucky have the opportunity to work with them.

References

[Benton. A(2017)] Mitchel. M Hovy. D Benton. A. 2017. Multitask learning for mental health conditions with limited social media data. *EACL* http://www.aclweb.org/anthology/E17-1015.

[Bingel. J(2017)] Sogaard. A Bingel. J. 2017. Identifying beneficial task relations for multi-task learning in deep neural networks https://arxiv.org/pdf/1702.08303.pdf.

[Caruana(1997)] Rich Caruana. 1997. Multitask learning 1. https://www.cs.cornell.edu/ caruana/mlj97.pdf.

[Kerinec E.(2018)] Sogaard A. Braud C. Kerinec E. 2018. When does deep multi-task learning work for loosely related document classification tasks? *ACL* http://aclweb.org/anthology/W18-5401.

[Luong. M(2016)] Le. Q Sustskever. I Vinyals. O Kaiser. L Luong. M. 2016. Multitask sequence to sequence learning. *ICLR* https://nlp.stanford.edu/pubs/luong2016iclr$_multi.pdf$.

[Martinez. H(2017)] Plank. B Martinez. H. 2017. When is multitask learning effective? semantic sequence prediction under varying data conditions. *EACL* http://www.aclweb.org/anthology/E17-1005.

[Mou. L(2016)] . et al Mou. L. 2016. How transferable are neural networks in nlp applications? *Arxiv* https://arxiv.org/pdf/1603.06111.pdf.

[Ratner. A(2019)] Hancock. B Dunnmon. J Sala. F Pandey. S R. C Ratner. A. 2019. Training complex models with multi-task weak supervision. *AAAI* https://arxiv.org/pdf/1810.02840.pdf.

[Ratner. A(2018)] Hancock. B R. C Ratner. A. 2018. The role of massively multi-task and weak supervision in software 2.0. *ACL* https://ajratner.github.io/assets/papers/software$_{2}mt_{v}ision.pd$

[Ruder(2017)] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks .