Homework 1: Tranfer Learning, Multi-Task Learning, and Few-shot Learning

Rifaa Qadri and **Josh Davis** and **Benjamin Wu** and **Shelvin Pauly** and **Michel Temgoua**
UMD
Computer Science Department
8125 Paint Branch Dr, College Park, MD

Abstract

This document will discuss the experiments and results found in Homework 1. We first explore domain and task adaptation techniques in Transfer learning. Then, we implement various multi-task learning weighting schemes and test few-shot transfer techniques.

1 Transfer Learning

1.1 Zero-Shot Transfer Baselines

We import MultiNLI matched and extract the training dataset with genre types 'Telephone' and 'Fiction' as the source distribution. We also extract validation-matched dataset with the genre types 'slate' and 'travel' for the target distribution. We begin by finetuning a pre-trained BERT encoder on the source distribution and evaluate on the target distribution. Note: DistilBertForSequenceClassification identifier use for base. An accuracy of 74.6 was achieved and our model state is saved for the next step.
We further finetune this model by training 10% of the target-domain training set (where the genre is of type 'slate' and 'travel') and evaluate the same dataset previously evaluated (our target distribution). We notice a slight increase in our accuracy to 78.37. We can therefore conclude that data points from our source domain are not all equally as important for the target domain transfer but rather, some add only noise and can overfit our model. In fact, when experimenting with adapting the domain in which we train and evaluate on, our accuracy suffers. Thus, methods that rank the source data and select only what is relevant for training would result in a more predictive model. We will experiment with this idea in the following experiment.

1.2 Reweighting Algorithm

With our baseline models trained and evaluated, we continue by exploring a domain adaptation technique to finetune a new pre-trained BERT encoder. The goal is to minimize

$$E_{pT_{x,y}}[L(f_\theta(x), y] = E_{pS_{x,y}}[\frac{pT_{x,y}}{pS_{x,y}} L(f_\theta(x), y]$$

We will first need to finetune a new pretrained BERT encoder that predicts the (binary) label of the input sentence of either the source or target domain data. In other words, our model aims to predict $p(targetdomain|x)$ and $p(sourcedomain|x)$. This requires the existence of some similarity between the two domains, and our model aims to bridge that gap. By using a similar framework as the BertForSequenceClassification, we train our model with a learning rate of $2e^-6$, a batch size of 32, and achieve an accuracy of 98-99 % with a single epoch. We save our model and the importance weights in the .bin file (418M). Next, we finetune a new model for our original source and target distribution. We unpack our saved pre-trained logits during training and apply a softmax to retrieve the corresponding probabilities $p_S(s)$ and $p_T(s)$. We can then apply element-wise multiplication with our loss function, and finally perform our own reduction bu taking the mean prior to the next back-prog step. By performing training in this matter, we are making an educated selection of our training samples.
Accuracy with (epochs=1,batch=8, rate=2e-5): 35.1% Accuracy with (epochs=3,batch=16, rate=5e-5): 55.1%

Clearly, we can see that our accuracy did not ameliorate or 'learn' much compared to our baselines, and it is likely that a negative transfer happened. By hypothesis, our algorithm relies on the existence of a single hypothesis with low error, which is not the case with our datasets. In order to perform as well or better than the base, we need the source distribution to cover the gap in the domain in order for the training data to be useful in learning our target. One way of accomplishing this is by creating the needed shared support through a source encoder and by matching the features at a population level.

1.3 Task Adaptation

As a comparison against domain adaptation, we also investigate the performance of task adaptation. In this case, the pretraining task is machine reading comprehension (MRC) and the target task is natural language inference (NLI), the same target as in our domain adaptation experiment. We begin with a new pretrained Hugging Face DistilBertForQuestionAnswering model and train it on the entirety of SQuAD-2.0, achieving an average F_1 score of 0.70 and an average exact match rate of 0.70. We then use the backbone of that model to initialize the backbone of a DistilBertForSequenceClassification model, where the new sequence classification head is initialized randomly. The whole model is then trained on 10% of the available 'slate' and 'travel' data in the MultiNLI dataset, as in our domain adaptation experiment, resulting in a final accuracy of 69% when validated against the 'slate' and 'travel' genres within the matched validation set.

1.4 Analysis

We implement a transferability metric, LEEP score, to evaluate the transfer performance of the models. Scores are computed for all experiments, including the zero-shot transfer and continued finetuning baselines. -0.2982645641737314 recorded from base 0. -1.069369763021558 recorded for reweighted. We can note that while the accuracy metric gives us more insight into how well we performed on the validation set, the LEEP score gives more insight into the transfer of knowledge between tasks being performed. According to our results, the task adaptation performed overall better than domain adaptation.

To outperform the task and domain adaption technique performed above, we can combine the two by fooling a domain classifier while learning a feature extractor that can map both the source and the target input to the same feature space. This allows to minimize the P(domain|f(x),y) and train a classifier on our source and target while using the same reweighing scheme to reweigh the source domain samples. Because this creates shared support and considers which samples are most relevant to us, it would outperform the models above.

2 Multi-Task Learning for MRC

2.1 Implementation

Here we describe how we implemented the necessary training and data processing required to complete the assignment.

2.1.1 BERT for Named Entity Recognition

Our implementation of BERT for the Named Entity Recognition (NER) task with WikiNEuRal is straightforward. We specifically use the DistilBERT model for all tasks to save development time with minimal sacrifice of accuracy. With one training epoch on the entire English training set of WikiNEuRal, we get validation accuracy of 98.59%.

2.1.2 Multi-task Training on NLI and NER

Our Natural Language Inference (NLI) to NER sequential, vice versa sequential, and simultaneous implementations are all expressed in a single script which relies on the provided TaskSampler code. We use the multi_nli data as required for the assignment, and once again are using the DistilBERT pre-trained model as a starting point. To get training on just one of the two training tasks, we set the TaskSampler weights a one-sided configuration, or in other words, to 1.0 for the task to train on and 0.0 for the other. To get sequential training from one task to the other, we simply set a one-sided configuration, and then flip to the opposite one-sided configuration halfway through the training. For example, for evaluating a sequential training starting with NLI and going to NER, we set task weights to 1.0 for NLI and 0.0 for NER at first, and then halfway through switch to 1.0 for NER and 0.0 for NLI. We choose a number of updates equal to the minimum of the lengths of the NER and NLI datasets. The main difficulty of this task was setting up training and evaluation loops without the use of a HuggingFace trainer, which was not compatible with using TaskSampler to draw batches.

We also set up static and dynamic weighting systems as needed to test how manipulating TaskSampler weights affected performance. In addition to manipulating the static weights of NLI and NER at start with TaskSampler, which affects how likely each task is to be sampled in simultanous training, we also devise a system to test static weighting configurations with the two orderings of sequential training. Rather than adjusting TaskSampler

	NLI		NER		MRC	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
$(0.5, 0.5, 0)$	0.708		0.983	0.870	0.951	4.087
Weight 2 $(0.75, 0.25, 0)$	0.735		0.981	0.851	1.179	3.867
Weight 3 $(0.25, 0.75, 0)$	0.669		0.985	0.885	0.294	3.843
Weight 4 $(1, 0, 0)$	0.704		0.0750	0.0162	0.699	3.653
Weight 5 $(0, 1, 0)$	0.3433		0.983	0.868	0.539	3.831
Weight 4 with 3 epochs $(1, 0, 0)$	0.711		0.161	0.0117	1.398	4.493
Weight 5 with 3 epochs $(0, 1, 0)$	0.327		0.982	0.867	1.061	4.271

Table 1: Simultaneous with static weights

weights, which need to be fixed to one-sided settings to get sequential training, we simply adjust the portion of the training dataset that is made available to the TaskSampler, making the NLI or NER training in the sequence relatively longer or shorter as desired.

For dynamic, we implement the Dynamic Weight Average (DWA) as described. We adjust the formula slightly, omitting the term T in the numerator to ensure the values sum to 1 as needed for sampling weights. For all experiments, we use a temperature (σ) value of 2, which is what the authors of the DWA metrics employ in their original paper (Liu et al., 2019).

2.1.3 Few-Shot Transfer to MRC

To do a few-shot transfer to Machine Reading Comprehension (MRC), also known as Question Answering (QA) using the SQuAD v2 dataset, we train a question answering head on top of the pretrained backbone created in the previous section. This training is few-shot in that we freeze the backbone parameters to prevent overfitting and use only 10% of the available MRC data in SQuAD v2. Once again, the most challenging aspect of this implementation was correctly setting up the no-trainer evaluation loop, which required using some unfamiliar post and pre-processing functions to prepare the data for training and evaluation.

2.2 Experiments

Here we describe various experimental campaigns conducted and analyze the results.

2.2.1 Simultaneous with Static Weights

We begin our interleaved experiments by randomly retrieving our batch from either NER or NLI and we experiment with various static weights. The following results are recorded: (See Table 1)

From the above, we can see the impact of one task over the other on the performance of MRC. We accomplish the highest F1 score of 4.087 on MRC by training NLI and NER equally. We further test the performance of our model in training one of the two tasks for an extended period of time. The two experiments using three epochs are shown in table 1.

Indeed, training T1 for an extended period of time (3 epochs) results in a better performance of MRC. We achieve the highest f1 score thus far, of 4.493. However, we also note that training for an extended period of time on both tasks simultaneously results in our model starting to overfit and performing worse across all validation sets.

2.3 Simultaneous with Dynamic Weights

We can observe that using dynamic weights under simultaneous training does not improve our results by much. Indeed, we conduct the experiments found in table 2. Once again, we see the highest correlation between the NLI and MRC tasks. Our highest f1 score under those parameters was

	NLI		NER		MRC	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
$(0.5, 0.5, 0)$	0.73		0.98	0.871	1.0275	4.1374
Weight 2 $(0, 1, 0)$	0.7134		0.9836	0.8746	0.9433	4.1237
Weight 3 $(1, 0, 0)$	0.7287		0.9836	0.8746	1.1876	4.3321

Table 2: Simultaneous with dynamic weights

4.3321 and was achieved by relying more on the NLI Training than the NER. Moreover, we observe that we have done overall better across all validation sets with dynamic weights. Intuitively, this makes sense as we are ensuring that one task's loss does not decrease at a slower rate than another.

2.4 Sequential Training NER -> NLI

We begin our sequential experiments with training on NER before switching to train NLI. The following weights have been experimented on (Please see table 3).

Our highest f1 score (of 4.43) was achieved on MRC by continuously relying heavily on NER rather than NLI (by setting our weight to .25,.75,0.1). Yet, we note that the difference in our accuracy of MRC is not much higher than by relying on NLI instead (.75,.25,.1) resulting in 4.28 F1 score. This further confirms our hypothesis that even when we began training on NER, NLI proves to be more useful to our target.

2.5 Sequential Training NLI -> NER

Next, we will observe the performance of MRC when we begin with training NLI prior to training NER using the following weight schemes (please see table 4).

Under those parameters, we notice that our highest f1 score of 4.349 was achieved when using both NLI and NER equally. We further investigate this fact by incrementing the number of epochs. Indeed the f1 score of 5.32 was achieved when training for a longer period of time on [1,1,0.1]. Since we have previously seen how well NLI transfers to MRC, it does make sense that our accuracy continues to increase the more we include NLI data. Indeed, the NLI dataset proved to be most useful in our experiments. In fact, the best F1 score of 11.19 was achieved on MRC when we trained 0.1 of NLI dataset and none of NER.

3 Contributions

The following portions / experiments have been completed by the following members:
Transfer Learning Baselines + Reweighting Algorithm: Rifaa Qadri
Transfer Learning Task Adaptation + LEEP Score: Benjamin Wu
Multi-Task Script: Josh Davis, with some contributions from Shelvin Pauly, Michel Temgoua and Rifaa Qadri
Multi-Task Simultaneous Experiments with Dynamic weights: Rifaa Qadri and Josh Davis
Multi-Task Simultaneous Experiments with static weights: Rifaa Qadri
Multi-Task Sequential Experiments: Josh Davis and Rifaa Qadri
Report Contributions:
Transfer Learning analysis: Rifaa Qadri
Multi-Task Learning: Josh Davis and Michel Temgoua
Multi-Task Experiments Report: Rifaa Qadri.

References

Shikun Liu, Edward Johns, and Andrew J Davison. 2019. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880.

	NLI		NER		MRC	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
$(1, 1, 0.1)$	0.7289		0.9681	0.7853	0.3201	4.2178
Weight 2 $(0.75, 0.25, 0.1)$	0.6438		0.9567	0.6775	1.0023	4.2805
Weight 3 $(0.25, 0.75, 0.1)$	0.7030		0.9539	0.6736	0.9770	4.4345
Weight 1 with three epochs $(1, 1, 0.1)$	0.7284		0.9648	0.7741	0.5559	3.8542

Table 3: Sequential NER -> NLI

	NLI		NER		MRC	
	Accuracy	F1	Accuracy	F1	Accuracy	F1
$(1, 1, 0.1)$	0.476		0.9830	0.8697	1.0865	4.3495
Weight 2 $(0.75, 0.25, 0.1)$	0.4881		0.9773	0.8167	1.0275	4.1653
Weight 3 $(0.25, 0.75, 0.1)$	0.5396		0.9823	0.8635	1.0612	4.0059
Weight 1 with three epochs $(1, 1, 0.1)$	0.537		0.982	0.86	2.66	5.326

Table 4: Sequential NLI -> NER