Reproducing PACE: Learning Effective Task Decomposition for Human-in-the-loop Healthcare Delivery

Ilya Andreev and Raj Krishnan
{iandre3, rajk3}@illinois.edu

Group ID: 62
Paper ID: 181, Difficulty: Hard
Presentation link: https://mediaspace.illinois.edu/media/t/1_7ls4cqgl
Code link: https://github.com/andreev-io/PACE

1 Introduction

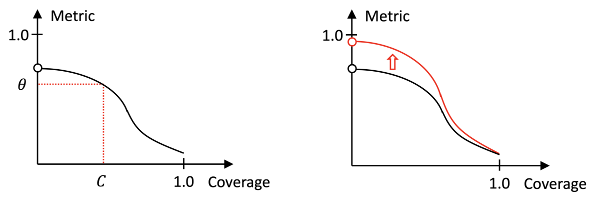
In the healthcare domain, it is paramount that neural systems don't make critical mistakes. As such, it is a common approach in the field to build neural classifiers that are only applied to easier tasks, while harder tasks are left for human professionals to handle. This means that there is a need to build specialized models that are able to classify easy tasks very accurately rather than classify all tasks somewhat accurately.

The paper we are reproducing in this work, PACE: Learning Effective Task Decomposition for Human-in-the-loop Healthcare Delivery (Zheng et al., 2021), develops a method of improving model performance on easy tasks. Zheng et al. demonstrate their method by applying it to the mortality prediction task based on ICU admission information taken from the MIMIC-III dataset (Johnson et al., 2016).

2 Scope of reproducibility

The PACE framework makes task decomposition more attainable via its two-level optimization approach which we describe in section 3.4. The framework is evaluated based on the Area Under the Curve (AUC) of the Receiver Operating Characteristic (ROC) for a given value of coverage, where coverage is defined as the share of tasks accepted by the model for classification.

An improvement in the AUC-Coverage graph is seen when we achieve a higher value of AUC for a given value of coverage. This is visualized in Figure 1 (Zheng et al., 2021). The focus of the PACE framework is to improve AUC for lower values of coverage without decreasing it for higher coverage values.



(a) Metric-Coverage plot (b) Aim of PACE

Figure 1: A Metric-Coverage plot and the aim of PACE. AUC is the Metric in our replication.

2.1 Addressed claims from the original paper

We will be verifying the following claims in this paper:

1. Our first goal is to replicate the claim that the PACE framework applied to the underlying RNN model (section 3.1) achieves an AUC-Coverage plot shifted upwards as compared to the same model trained without the PACE framework, i.e. without Self-Paced Learning and without binary cross-entropy loss modifications explained in section 3.4.

2. Our second goal is to reproduce the ablation study that claims that even without the binary cross-entropy loss modifications (section 3.4), the same model produces an AUC-Coverage plot shifted upwards when trained with Self-Paced Learning as compared to when trained without Self-Paced Learning.

3. Our third goal is to reproduce the ablation study where we confirm whether the model trained with the modified binary cross-entropy loss yields an AUC-Coverage plot shifted upwards as compared to a model trained with regular binary cross-entropy loss. No Self-Paced Learning is used in this claim.

3 Methodology

3.1 Model descriptions

The model in question faces a binary prediction task, namely mortality prediction based on a patient's fixed-length time series input. Every entry in a patient's time series is a set of measurements taken for the patient in a 2-hour window during the first 48 hours of their ICU stay.

The model carrying out the task is a simple single-layer unidirectional Recurrent Neural Network (RNN) with a Gated Recurrent Unit (GRU) (Cho et al., 2014) with 32 as the hidden vector size. The RNN's final hidden state is fed into a fully-connected linear layer with a single value as the output. The sigmoid activation function is applied to the output of the linear layer, at which point a value greater than 0.5 is considered to predict mortality, whereas any smaller value is not.

3.2 Data description

Similar to the source paper, we use the MIMIC-III dataset (Johnson et al., 2016) to perform the mortality prediction task. Zheng et al. (2021) also evaluate their framework on a private longitudinal Electronic Medical Record dataset from the National University Hospital (NUH) in Singapore. We do not use the NUH dataset in our replication. To pre-process MIMIC-III data, we use the MIMIC-III extraction suite (Wang et al., 2020).

The extraction suite cleans up the provided MIMIC-III tables and splits each individual patient's readings into separate time series. While Zheng et al. (2021) extract 710 features for 52,665 tasks, we limit ourselves to 104 features and 15,591 tasks. We split these tasks into a validation dataset with 1,404 tasks with 125 positive cases and a test dataset with 1,560 tasks with 129 positive cases. The rest of the tasks form the training dataset, which we randomly oversample using Lemaître et al. methodology to bring its size up to 23,062 tasks with 11,531 positive cases, among which 1,096 positive cases are unique.

3.3 Hyperparameters

The hyperparameters are set to match the values used in Zheng et al. (2021) where possible. For parameters not provided by Zheng et al., we choose the best values we can obtain after experimenting with a few arbitrary options.

Parameter	**Value**
Tolerance	0.001
Patience	10
Epochs used	30 to 100
Warm-up epochs	1
Learning rate	0.001
Regularization	L2
Θ	0.0003
$N_{initial}$	16
λ	1.3
γ	0.5

Table 1: Hyperparameters

- Θ is a regularization coefficient.
- $N_{initial}$ is used in task selection for macro-level optimization (SPL).
- λ is a factor by which N is decreased after every epoch.
- γ is used in $BCELoss_{modified}$ for micro-level optimization.

3.4 Implementation

To reproduce Zheng et al. (2021), we use our own code implementation using PyTorch (Paszke et al., 2019): https://github.com/andreev-io/PACE. We manually implemented two central ideas of the paper: macro- and micro-level optimizations.

Our macro-level optimization adaptively selects the tasks on which to train the model in any given epoch by only considering tasks i for which

$$\text{BCELoss}_{\text{modified}}(x_i, y_i) < \frac{1}{N}$$

where $\text{BCELoss}_{\text{modified}}(x_i, y_i)$ is the binary cross-entropy loss for input x_i with ground truth label y_i, modified as discussed in the next paragraph. N is a variable initialized to $N_{initial}$ and decreased by a factor of λ after every training epoch. This optimization is a form of Self-Paced Learning (Kumar et al., 2010), which ensures that the model starts by training on easy tasks first and is only later trained on more difficult tasks.

On the micro-level, we apply a modification to the binary cross-entropy function to assign more weight to correctly predicted tasks. This is done by making it so the absolute value of the derivative of the modified cross-entropy function is higher for correctly predicted tasks as compared to the standard cross-entropy loss. In our implementation, we use

$$\text{BCELoss}_{\text{modified}}(x_i, y_i) = \\ = -\frac{1}{\gamma} \cdot y_i \cdot \ln \sigma(\gamma \cdot o_i) - \frac{1}{\gamma} \cdot (1 - y_i) \cdot \ln (1 - \sigma(\gamma \cdot o_i))$$

where o_i is the single-value output of the fully-connected linear layer of the model given input x_i, γ is a hyperparameter set to 0.5, and σ is the softmax activation function. The derivative of $\text{BCELoss}_{\text{modified}}$ loss with respect to o_i is $\sigma(\gamma \cdot o_i) - 1$.

The loss function also includes an L2 regularization summand multiplied by the Θ coefficient.

3.4.1 Training

Warm-up: Regardless of whether SPL is used, we first train the model on all tasks for the warm-up number of epochs, which is set set to 1. After this, if SPL is used, easy tasks are picked based on hyperparameter N as described in section 3.4. N is decreased by a factor of λ for every epoch following the warm-up epochs.

Early Stopping: We train the model for at least 30 epochs, at which point we checkpoint the model after every epoch. If after an epoch the decrease in loss on the validation dataset is less than the tolerance hyperparameter, we do not checkpoint the model and decrease the patience counter instead; once patience decreases to 0, we retrieve the last checkpointed model and stop training. If the validation loss decrease exceeds our tolerance, we checkpoint the model again and reset the patience counter. Similar to Zheng et al., we place an upper limit of 100 epochs on training, although we do not get close to this number in practice.

Optimizer: We use the Adam optimizer (Kingma and Ba, 2014) for training.

All implementations share the same mechanism for early stopping and the same optimizer. They also share the same warm-up code, which has no effect when SPL is not used.

3.5 Computational requirements

We perform all the necessary data pre-processing for the MIMIC-III dataset and model training and validation on the University of Illinois at Urbana-Champaign Hardware-Accelerated Learning (HAL) cluster (Kindratenko et al., 2020). We used four IBM POWER9 2.4GHz CPUs for data pre-processing and model development and one

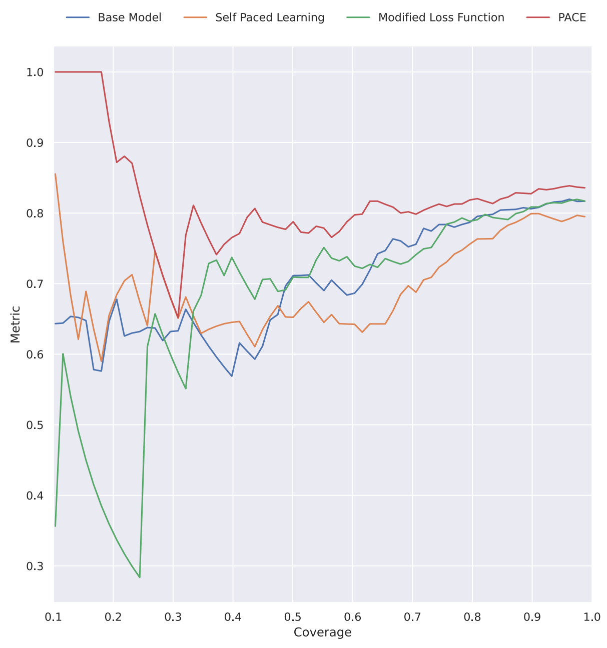


Figure 2: Metric-Coverage for 4 resulting models

5120-core Nvidia V100 16 GB HBM 2 GPU for training, testing, and validation.

The time spent training any of the models on this system is always within 10 minutes, which is well below our initial estimate of 1 hour for training. This is partially affected by early stopping, as we terminate the training process once we deem the training ineffective.

4 Results

We have implemented and evaluated the following four models:

1. RNN trained with SPL and the modified BCELoss.
2. RNN trained with SPL.
3. RNN trained with the modified BCELoss.
4. RNN trained without SPL and the modified BCELoss.

In our evaluation, we use the test dataset with 129 positive tasks and 1,560 tasks total. We run each model against the dataset and sort the predictions based on decreasing prediction confidence, where confidence on task i is defined as $0.5 + |\sigma(o_i) - 0.5|$.

This way any given value of coverage corresponds to an index of an entry in the list of predictions made, which in other words represents a confidence percentile. We calculate the AUC ROC for every such percentile starting from 0.1 and show it in figure 2. We ignore coverage values from 0 to 0.1 because just like Zheng et al. we have observed the performance at low coverage to fluctuate excessively.

Below we discuss the general trends seen in the results and compare them with the trends in the original paper. We do not compare specific absolute values as there are pre-processing steps which are not publicly shared in the original paper.

4.1 Claim 1: RNN trained with SPL and modified BCELoss is better than the baseline RNN

Much like Zheng et al. (2021), we find that model 1 outperforms the baseline for any value of coverage. Most importantly, we see that the biggest performance improvement is seen at lower values of coverage, meaning that the model focuses on classifying easy tasks well and with high confidence. This confirms that PACE is a successful framework in that it achieves the main goal illustrated in Figure 1.

4.2 Claim 2: RNN trained with SPL is better than the baseline RNN

We observed that model 2 performs marginally better than the baseline on low values of coverage and marginally worse than the baseline on high values of coverage. While this result confirms our expectation that SPL makes the model perform better on easier tasks, it is different from what was reported in Zheng et al. (2021), where an SPL-based model outperformed the baseline for all values of coverage, although by not as much as model 1.

4.3 Claim 3: RNN trained with the modified BCELoss is better than the baseline RNN

Unlike what was reported in Zheng et al. (2021), we found that a model trained purely with the modified BCELoss without SPL did not perform decidedly better than the baseline. We saw that model 3 performed worse than the baseline at low values of coverage, slightly better than the baseline at moderate values of coverage, and similarly to the baseline at high values of coverage.

5 Discussion

While we did not observe the results that Zheng et al. (2021) claim to have found for claims 2 and 3, we could replicate the results for claim 1, which is arguably the central piece of the work of Zheng et al. (2021) and demonstrates the effectiveness of the PACE framework. The framework considerably improves the performance for easy tasks, while showing a slight improvement over the baseline model for harder tasks.

5.1 What was easy

We approached the work strategically, implementing the framework end-to-end, albeit with mistakes and incomplete data. We then iterated on the end-to-end implementation and brought it to completion. This made it easy to improve the different parts in parallel, and ensured that we always had a working pipeline. Had we focused on implementing any specific part of the work perfectly before starting to work on subsequent parts, we would have failed to create a successful replication study.

5.2 What was difficult

We ran into issues preparing the data for use in the model. Our initial model was built using the MIMIC benchmark framework (Harutyunyan et al., 2019), but was limited to 17 variables used in the framework. It proved relatively difficult to extend this to work with more variables and led the model to consistently underperform compared to the original paper. We switched to using the data provided by Wang et al. (2020) as a base, which allowed us to replicate the results in the original paper.

5.3 Recommendations for reproducibility

Our first recommendation is to use a standardized data extraction pipeline or to publish an open source pipeline for data pre-processing. This allows future research on a similar topic to provide meaningful comparison points rather than requiring duplicated work for data extraction and pre-processing.

Secondly, we feel that it would be extremely valuable to post the source code used to build the system on a public platform, which would let other researchers build on the work. If this is not possible, an alternate approach could be an accompanying technical post which details the components of the system, the values of all hyperparameters used, and any specific implementation details such as the oversampling strategy and regularization function.

6 Communication with original authors

While we did not directly communicate with the authors, here is the list of questions we would ask if we could:

- How were the hyperparameters chosen?
- What data pre-processing steps were taken to retrieve 710 features for 52,665 prediction tasks
- Is there any other work being done related to the PACE framework?

References

Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches.

Hrayr Harutyunyan, Hrant Khachatrian, David C. Kale, Greg Ver Steeg, and Aram Galstyan. 2019. Multitask learning and benchmarking with clinical time series data. *Scientific Data*, 6(1):96.

Alistair E. W. Johnson, Tom J. Pollard, Lu Shen, Liwei H. Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. 2016. Mimic-iii, a freely accessible critical care database. *Scientific Data*, 3(1):160035.

Volodymyr Kindratenko, Dawei Mu, Yan Zhan, John Maloney, Sayed Hadi Hashemi, Benjamin Rabe, Ke Xu, Roy Campbell, Jian Peng, and William Gropp. 2020. *HAL: Computer System for Scalable Deep Learning*, page 41–48. Association for Computing Machinery, New York, NY, USA.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization.

M. Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc.

Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Shirly Wang, Matthew B. A. McDermott, Geeticka Chauhan, Marzyeh Ghassemi, Michael C. Hughes, and Tristan Naumann. 2020. Mimic-extract: A data extraction, preprocessing, and representation pipeline for mimic-iii. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, CHIL '20, page 222–235, New York, NY, USA. Association for Computing Machinery.

Kaiping Zheng, Gang Chen, Melanie Herschel, Kee Yuan Ngiam, Beng Chin Ooi, and Jinyang Gao. 2021. *PACE: Learning Effective Task Decomposition for Human-in-the-Loop Healthcare Delivery*, page 2156–2168. Association for Computing Machinery, New York, NY, USA.