Empirical Study of Neural Network Optimizers and Schedulers

Avinash Joshi
bwaseous@uw.edu
AMATH 515 Wi 24
8 March, 2023

Abstract

The proliferation of machine learning with the advent of large scale neural network (NN) architectures such as transformers highlights the necessity for robust optimization algorithms and learning-rate [LR] (a.k.a. step-size) schedulers to minimize training time and model hyper-parameter tuning while maximizing model performance. Three stochastic gradient descent algorithms (`AdamW`, `rmsprop`, and `Adagrad`) and three learning-rate schedulers (cosine annealing with restarts, linear LR decay, and polynomial LR decay) were empirically tested against three machine learning (ML) tasks — tabular data binary classification, named entity recognition (NER), and image classification. The combiniation of `AdamW` as the optimizer and either linear decay or cosine with restarts as scheduler continually produced the best results for all three tasks indicating it's robustness while `AdaGrad` performed the worst in all trials regardless of LR scheduler.

1 Introduction and Overview

The terms neural network (NN) and machine learning (ML), once a term relegated to statistics lectures and optimization seminars, have exploded into the broader cultural zeitgeist in the last decade. Having penetrated every facet of academic, professional, and lay life, the advent of large language models (LLMs) like OpenAI's GPT-4 [1] perform complex tasks such as image classification, data prediction, and natural language processing (NLP). This explosion in NN usage for varied applications has been quietly pushed along by the innovation of high performing and versatile stochastic gradient descent (SGD) algorithms and learning rate (LR) schedulers that, in conjunction with advancements in computing hardware via graphics processing units (GPUs), allow for larger-than-ever models to be reliably trained faster-than-ever.

For enterprise and research applications, fast model training times and high model performance are minimum requirements for using NNs in actual applications. A simple question

comes from these criteria: what is the "best" combination of stochastic gradient descent algorithm and learning rate scheduler for a wide variety of machine learning tasks using neural networks? To narrow the seemingly endless variety of SGD algorithms and LR schedulers, I will test three SGD algorithms and three LR schedulers that are readily implemented in machine learning libraries (`PyTorch`[14] and `transformers`[18]) in the programming langauge `python`: `AdamW` [12], `rmsprop` [7], and `AdaGrad` [5] for SGD algorithms; and, cosine annealing with restarts [11], linear decay, and polynomial (quadratic) decay for LR schedulers. The three ML tasks that these optimizers and schedulers will be tested against are a tabular binary classification on credit default [19], named entity recognition (NER) on Reuters' articles [16], and image classification on pigmented skin lesions [17].

In Section 2, I will discuss stochastic gradient descent, the theoretical implementations of the three stochastic gradient descent algorithms, and the learning rate schedulers; in Section 3, the three different ML tasks, the data pre-processing for the three selected tasks, and the metrics related to each; in Section 4, how the schedulers and optimizers were implemented in `python`; in Section 5, the computational results and a discussion of those results; and, finally, in Section 6, a conclusion of the results, the validity of the experiments, and further directions of inquiry.

2 Theoretical Background

In a supervised learning task, stochastic gradient descent (SGD), much like gradient descent (GD), operates by training a predictor or model, f, that, given an input to the predictor, x_i, which may be tensor-dimensional, approximates some output, y_i, given trainable parameters inherent to the predictor, θ,

$$y_i = f(x_i; \theta).$$

A metric, loss, or error, E, is computed given the correct output, y_i, and the predicted output, $\hat{y}_i$, and this loss is used to change the parameters, θ, in order to reduce the loss/error or increase/decrease some metric given by some function $\mathcal{L}$,

$$E_i = \mathcal{L}(x_i, y_i; \theta).$$

The predictor at time-step t, f_t, then minimizes $\mathcal{L}$ given inputs x_i and outputs y_i. These parameters θ, are then updated via gradient descent (where $g_t = \nabla_\theta \mathcal{L}_t(x; \theta_t)$, the gradient with respect to the parameters θ of the loss of predictor f_t at time-step t) with some sufficient step-size or LR parameter η,

$$\theta_{t+1} = \theta_t - \eta g_t. \tag{1}$$

Unlike GD, SGD updates θ given a singular (or a small batch) of x_i, rather than the entire data [2]. This is done because the input data x, is assumed to be redundant or sparse, i.e., the data is repeated [7]. For SGD, θ can be updated per input data, x_i, dubbed online training, or by a mini-batch of random inputs, $x_t = \{x_i : i \in \mathbb{N}^0 \leq n\}$ where n is the length

of training data, and where x_t is N dimensional, the batch size,

$$\theta_{t+1} = \theta_t - \frac{\eta}{N} \sum_{i \in \mathbb{N}^0 \leq n} \nabla_\theta \mathcal{L}(x_i; \theta_t). \tag{2}$$

In order for SGD algorithms to converge, however, the LR parameter η, must be changed per iteration, i.e., $\eta = \eta(t)$. While Equation 2 can be used to train models, more sophisticated algorithms such as `AdamW`, `rmsprop`, and `AdaGrad` have been developed to increase the rate of empirical rate of convergence, relegating Equation 2 to the mantel of a foundational algorithm that is expanded upon by other techniques rather than something used professionally.

2.1 `AdamW`

The most extensively used algorithm when training neural networks is the `Adam` family of SGD algorithms, an adaptive gradient method. Adaptive gradient methods work by altering the gradients g_t at each time step depending on the previous gradients, g_t for $t = 1, \ldots, t-1$. `Adam` [8], and it's weight-decay regularized older brother, `AdamW` [12], employ a technique to update exponentially moving averages of the gradients, m_t, and squared-gradients, v_t, which are estimates of the 1^{st} moment (mean) and 2^{nd} moment (uncentered variance), respectively. These averages decay at a rate governed by $\beta_1, \beta_2 \in (0, 1]$ and are then bias-corrected away from zero, their initializations, into $\hat{m}_t$ and $\hat{v}_t$. `AdamW` expands upon it's predecessor by altering the gradient at the stepping scheme η_t rather than altering the "raw" gradient, g_t. `AdamW` is given in Algorithm 1.

Algorithm 1: `AdamW`. η_t the scheduled LR; $\beta_1 = 0.9$, $\beta_2 = 0.99$; $\lambda \in \mathbb{R} \geq 0$ the weight decay; $m_0, v_0 = \mathbb{0}$; $\epsilon = 10^{-8}$

for $t = 1, \ldots, T$ **do**

$g_t \leftarrow \nabla_\theta \mathcal{L}(x_t; \theta_{t-1})$

$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

$\theta_t \leftarrow \theta_{t-1} - \eta_t \left(\frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_{t-1} \right)$

end

2.2 `rmsprop`

`rmsprop` [7], a similar type of algorithm to `AdamW`, alters the gradient by the root of the mean square of the previous gradients, but, does not account for bias-correction like `AdamW`

does. Likewise, `rmsprop` does not encode any mean information, opting just for information about the variance of the gradients. Algorithm 2 details the theoretical implementation of `rmsprop`.

Algorithm 2: `rmsprop`. η_t scheduled learning rate; $\alpha = 0.99$; $v_0 = \mathbb{0}$; $\epsilon = 10^{-8}$

for $t = 1, \ldots, T$ **do**

$$g_t \leftarrow \nabla_\theta \mathcal{L}(x_t; \theta_{t-1})$$

$$v_t \leftarrow \alpha v_{t-1} + (1 - \alpha) g_t^2$$

$$\theta_t \leftarrow \theta_{t-1} - \eta_t \frac{g_t}{\sqrt{v_t} + \epsilon}$$

end

2.3 `AdaGrad`

Developed before `rmsprop` and `AdamW`, `AdaGrad` [5] is an algorithm designed for sparse gradients where the gradient g_t is altered at each step by the root of the sum of the previous gradients. Effectively, parameters θ with smaller variance will have larger learning rates while higher variance parameters will have smaller learning rates. Unlike `AdamW`, `AdaGrad` only encodes information about the model parameter's variance and not mean as well. Algorithm 3 details the theoretical implementation of `AdaGrad`. This (and most) implementations of `AdaGrad` are approximations to the sum of the outer products of all previous gradients ($\frac{1}{\sqrt{G_t}}$ becomes $G^{-1/2} = (\sum_{i=1}^{t} g_i g_i^T)^{-1/2}$), a costly yet powerful way to encode information about the previous gradients much like the Jacobian does for Newton's Method [5]. This paper's implementation is equivalent to taking the diagonal entries of the $G^{-1/2}$.

Algorithm 3: `AdaGrad`. η_t scheduled learning rate; $G_0 = \tau \geq 0$; $\epsilon = 10^{-10}$

for $t = 1, \ldots, T$ **do**

$$g_t \leftarrow \nabla_\theta \mathcal{L}(x_t; \theta_{t-1})$$

$$G_t \leftarrow G_{t-1} + g_t^2$$

$$\theta_t \leftarrow \theta_{t-1} - \eta_t \frac{g_t}{\sqrt{G_t} + \epsilon}$$

end

2.4 Learning Rate (LR) Schedulers

SGD algorithms, by the often nonlinear nature of the problem they solve, there is no guarantee that the initial learning rate η will allow the fixed point iteration of θ to converge,

and, if it converges, in a timely manner; therefore, by modifying η based on a metric's performance or scheduled based on a per-step basis (usually per epoch[1] or per mini-batch), the model's convergence and speed of convergence can be altered to better fit the geometry of the problem and the loss function.

2.4.1 Linear Decay

After a constant learning rate, the linearly decaying learning rate is one of the most basic of learning rate scheduling which simply multiples the initial learning rate η_0 by a linear interpolant that governs the linear rate decay, $\gamma(t)$. For T, the total number of scheduler iterations, the linear decay scheduler is given by the linear interpolant between a start factor γ_0 and the end factor γ_T for $t < T$ and γ_T for $t \geq T$,

$$\eta_t = \underbrace{\left(\frac{\gamma_T - \gamma_0}{T} t + \gamma_0\right)}_{\gamma(t)} \eta_0. \tag{3}$$

Note, `huggingface`'s linear decay scheduler functions as Equation 4 with $n = 1$ and T the number of training steps.

2.4.2 Polynomial Decay

Polynomial decay simply raises a decaying factor that decreases linearly to zero from 1 with each step to the nth power,

$$\eta_t = \underbrace{\left(1 - \frac{t}{T}\right)^n}_{\gamma(t)} \eta_0. \tag{4}$$

2.4.3 Cosine annealing (with restarts)

Cosine annealing [11] alters the learning rate by a cosine function between a minimum learning rate, $\eta_{\min}$, and η_0 (also called $\eta_{\max}$ in [11]),

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min})\left[1 + \cos\left(\frac{t}{T}\pi\right)\right]. \tag{5}$$

In this paper, $\eta_{\min} = 0$. This way, instead of monotonically increasing or decreasing during training, the model has opportunities to have variable speed training. The cosine function can be altered slightly in order to cut off the increasing portion of the cosine function via a restart by letting $t = t \bmod T$. Likewise, after each restart, T can also be modified by a parameter $T_{\text{mult}} > 0$ to alter when the restarts happen via $T = T \times T_{\text{mult}}$ when $t \bmod T = 0$ for $t > 0$.

[1] An epoch is how many times the model has been trained on a set of training data.

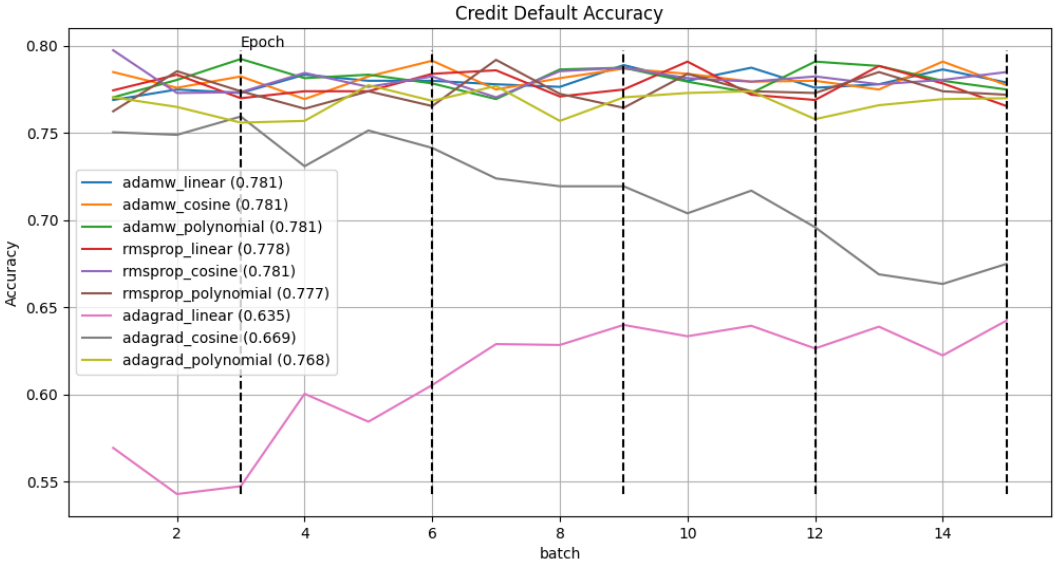

Figure 1: Test accuracy for the optimizers and schedulers on the credit default task. The labels in the legend correspond to the optimizer then scheduler and the total accuracy on the test set in parentheses. The `AdaGrad` methods performed below the `AdamW` and `rmsprop` methods.

3 Task Overview and Model Architectures

To test the these optimizers and schedulers, three unique datasets and their associated problem task were chosen based on the range of potential applications neural networks are applied to. For each problem task and dataset, the problem will be introduced, the techniques used to pre-process the datasets outlined, and the network architectures for each model used.

3.1 Tabular Data Binary Classification: Credit Default

One of the cornerstones of machine learning is binary classification based on tabular data. The problem is thus: given features (numerical, categorical, etc.), predict a binary output (0 or 1). To diagnose this common problem type with our chosen optimizers and schedulers, a credit default tabular dataset was chosen. Tracking 33 demographic and financial statistics of 30,000 Taiwanese credit card owners between in 2005, the "default of credit cards" dataset [19] attempts to predict whether these credit card owners will default the following month. Split into train and test splits of 24,000 and 6,000 individuals, respectively, a simple feed-forward neural network with 4 linear layers connected between each other by ReLU [6] activation functions, finally terminating in a sigmoid function to map the outputs $\hat{y}_i \in [0, 1]$. The layer dimensions, in order, are 33×256, 256×256, 256×128, and 128×1. The network architecture is not entirely important for this paper as it just serves as a empirical testing ground; likewise, no feature engineering was performed on the dataset besides that which was present when loaded from the Huggingface `datasets` repository ("imodels/credit-card").

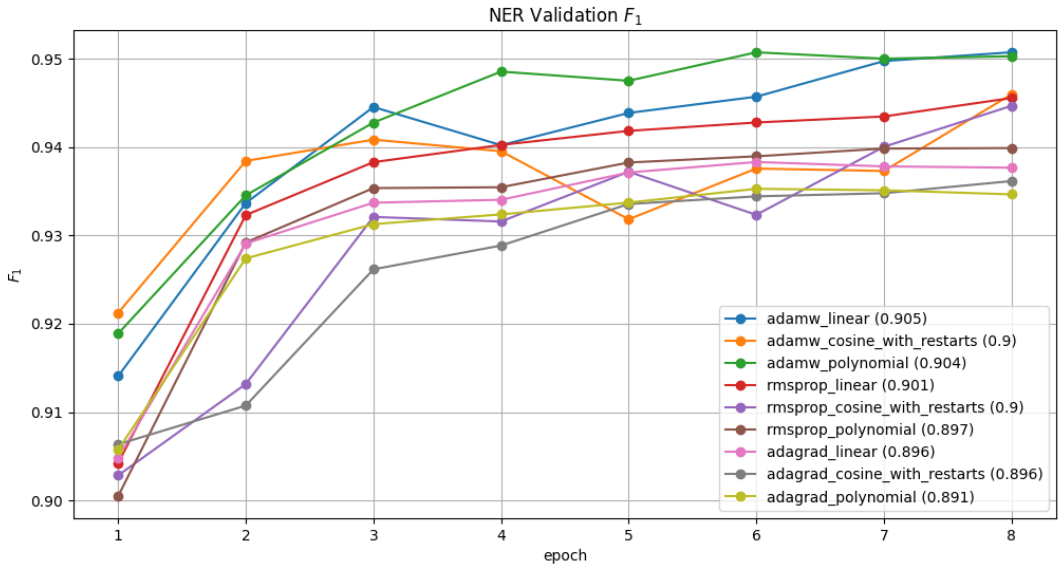

Figure 2: Validation F_1 on the CONLL2003 dataset. All optimizers and schedulers performed well (within a couple percentage points of each other), but `AdamW` consistently had higher F_1 training validation. The test F_1 is in parentheses.

For this type of problem, stepping the scheduler can be done per epoch or per batch, but this paper implements a per epoch stepping scheme.

For this problem, the model's performance was determined by accuracy, which I define as the percent of correctly predicted outputs $\hat{y}_i$ given the features x_i.

3.2 Named Entity Recognition (NER)

Named entity recognition (NER) is a common natural language processing (NLP) task where words in a sentence are classified as one of a number of possible classes [9]. Large language models such as the Bi-directional Encoder Representations from Transformers (BERT) family [3] of models, like `BERT` and `RoBERTa` [10], are trained to predict masked words in sentences given the context of the whole sentence and to predict whether one sentence precedes another in order to replicate and approximate the understanding of human language. This pre-trained model can then be adapted to NER by way of changing the "head" of the model. The core mechanics of the model stay the same, but a linear activation layer is added on as the final output of the model and trained for a specific task, like predicting whether a word in a sentence is one of set of specific classes. The model for this task used `distilroberta-base` due to it's significantly smaller size compared with it's sister, `RoBERTa`, at 83 million parameters compared to 125 million, while performing at a similar level. This not only reduces training time since the model is smaller, but can approximate a choice that a business might

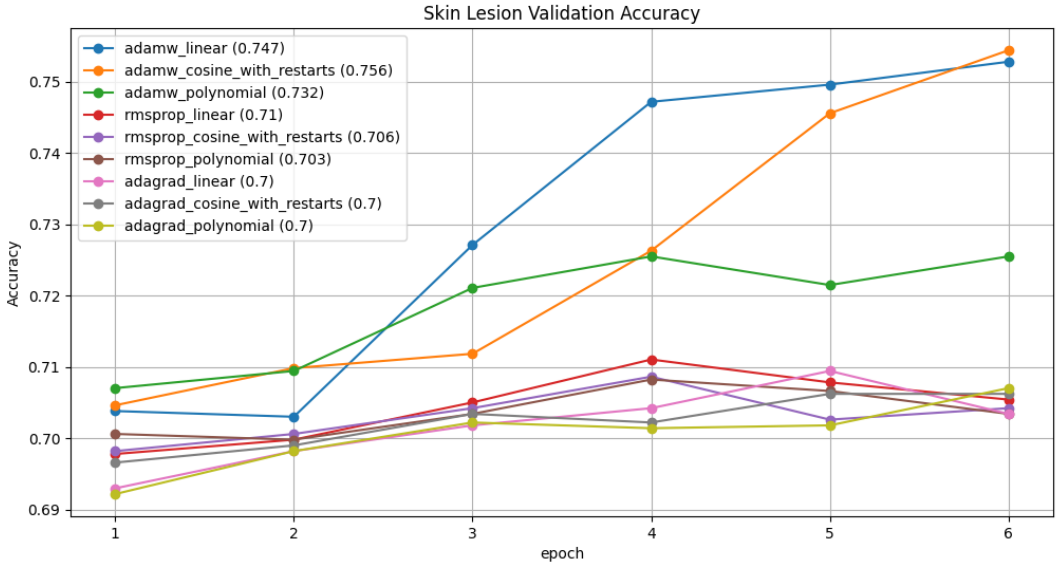

Figure 3: Validation accuracy for the HAM10000 dataset. `AdamW` methods performed many percentage points above `rmsprop` and `AdaGrad`, where both the inferior optimizers failed to learn little to anything about the problem after the fist epoch of training. In this complex task cosine annealing with restarts performed better than all other schedulers.

make in order to fit the model on less powerful devices.

The "CONLL2003" [16] English dataset was used for this task which is comprised of 20,744 sentences, split into train, validate, and test splits, from the Reuters news stories between August 1996 and 1997. In these sentences, the named entities "person" (PER), "organization" (ORG), and "miscellaneous entity" (MISC) are classified using BIO tagging. To track the model's performance, the F_1 metric which is a composite score of precision and recall,

$$\text{precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \qquad \text{recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}},$$

and

$$F_1 = 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

The only pre-processing done to this dataset was tokenizing the dataset which converts the input sentences into a numerical representation required for the model.

3.3 Image Classification: Skin Lesion Identification

For the last task, image classification is a computer vision task where images are classified into two or more classes based on a machine learning algorithm. While a relatively trivial task for humans, machine learning, and specifically the adoption of convolutions to encode complex relationships between pixels of the images, has drastically improved this pivotal part of computer vision [15]. Aided further by the development of the transformers architecture [18], models like `ViT`[4] can match or outperform convolutions neural networks (CNN). This paper employs the `MobileViTv2` model [13], a down-scaled vision transformer and CNN with about three million parameters on the "HAM10000" ("Human Against Machine with 10000 training images") [17] dataset. The "HAM10000" dataset contains 10,017 images of seven different types of pigmented skin lesions including actinic keratoses, basal cell carcinoma, "benign keratosis", etc. The dataset was split into train (about 9,580 samples), validate (about 2,490 samples), and test (about 1,290 samples) splits where the following transformations were applied randomly to the training images: random resize crop, a Gaussian blur, a color jitter, and a random rotation. Like Section 3.1, accuracy was used as the model's metric.

4 Algorithm Implementation

The implementation of the three SGD algorithms, `AdamW`, `rmsprop`, and `AdaGrad`, was fascillitated using `pytorch`'s implementation via `torch.optim.AdamW/RMSprop/Adagrad()` classes. Specific arguments described in Section 2, like LR, are passed to the class alongside the model's parameters via `model.parameters()`. For learning rate scheduling, the task outlined in Section 3.1 uses `pytorch`'s implemenetation of the three schedulers via `torch.optim.lr_scheduler` and following classes; `LinearLR` with $\gamma_0 = \frac{1}{3}$, $\gamma_T = \frac{1}{1000}$, and $T = 5$ epochs; `CosineAnnealingWarmRestarts` with $\eta_{\min} = 0$ and $T = 5$ epochs; and, `PolynomialLR` with $T = 5$ epochs and $n = 2$. For the tasks in Section 3.2 and 3.3, `transformers`'s learning rate scheduler were used and accessed through `transformers.get_scheduler`. For Section 3.2 and 3.3, the number of cycles was chosen to be 2, i.e., the cosine reset once.

At each mini-batch iteration, the batch was passed through the model, it's loss calculated using the loss function for each problem, and the gradients were back-propagated. For `transformers`, after each mini-batch the scheduler is stepped, but for my implementation in `pytorch`, the scheduler was stepped after each epoch. For the credit default dataset, 5 epochs were used with a batch size of 2000; for CONLL2003, 8 epochs and a batch size of 150; and, for HAM10000, 6 epochs with a batch size of 80.

		Scheduler		
		Linear	Poly	Cosine
SGD	`AdamW`	**0.781**	**0.781**	**0.781**
	`rmsprop`	0.778	0.777	**0.781**
	`AdaGrad`	0.635	0.768	0.669

(a) Credit default accuracy where 0.781 is the highest accuracy. For this task many different optimizers and schedulers performed optimally.

		Scheduler		
		Linear	Poly	Cosine
SGD	`AdamW`	**0.905**	0.904	0.900
	`rmsprop`	0.901	0.897	0.900
	`AdaGrad`	0.896	0.891	0.896

(b) NER F_1 where `AdamW` with linear decay performed the best with an F_1 of 0.905.

		Scheduler		
		Linear	Poly	Cosine
SGD	`AdamW`	0.747	0.732	**0.756**
	`rmsprop`	0.710	0.703	0.706
	`AdaGrad`	0.700	0.700	0.700

(c) Skin lesion classification accuracy where `AdamW` with cosine with restarts performed the best with an accuracy of 0.756

Table 1: Model performance for optimizer and scheduler combinations for each problem. The `AdamW` optimizer continually did the best between all three problems while the linear and cosine with restarts schedulers traded best performance.

5 Computational Results and Discussion

After manually hyperparemeter tuning the optimizer learning rates and scheduler parameters for each problem crudely, the problem metrics were collected and displayed in Table 1. For the tabular binary classification task with the Credit Default dataset, many different schedulers and optimizers did the best, but the `AdamW` optimizer tied all it's schedulers for best performance. For the NER task with the CONLL2003 dataset, all the schedulers and optimizers performed well possibly indicating that that smaller `distilroberta-base` model was the limiting factor in task performance; however, `AdamW` still performed the best this time with the linear scheduler. The most challenging of the tasks was the image classification task with the HAM10000 dataset. Here, the `AdamW` optimizer performed far-and-away better results than `rmsprop` and `AdaGrad` which both failed to improve their in-training test

accuracy. This time, the cosine with restarts scheduler performed the best with almost a whole percentage point better than than the next scheduler. During training, because of the cosine with restarts scheduler's periodic nature, the model initially fails to make headway on the problem, but quickly rises to the top of performance by the end. This is in contrast to the linear scheduler which at first fails to make progress, likely due to having a LR too large, but rapidly makes progress only to plateau at the end. The cosine's periodic nature, while initially failing to make major progress compared to other schedulers across the different problems, make's up for the comparatively smaller LRs at the end of training compared with the other schedulers.

Between all the optimizers, `AdamW` routinely performed the best regardless of scheduler while `AdaGrad` always did the worse. The polynomial scheduler comparatively performed the worse out of all the schedulers often having it's accuracy in training plateau earlier than other methods (the green line of Figure 3). This is likely due to the fact that the LR of the polynomial scheduler decays to 0 faster than the linear scheduler and is non-periodic unlike the cosine scheduler.

One important caveat must be made in relation to the training accuracy: the results are only accurate up to whole percentage points or the second digit of accuracy. If one were to retrain the models on the same dataset while setting the seeds of the various random number generators, as used by the `set_seed()` method, the model's performance are only accurate to the first decimal place and the second decimal place up to about one hundredth. Because of this uncertainty, the third task, image classification, and the stark contrast in performance between `AdamW` and the other two optimizers, clearly paints a clear picture about which optimizer to use. If one takes the hundredths place as fact, as well, either the cosine with restarts or linear scheduler are both appropriate choices. While the conclusion about using `AdamW` being superior can be accepted, whether cosine with restarts or a linear scheduler is better is undecided. Larger models trained longer with more sophisticated feature engineering and data pre-processing is required to completely suss out which scheduler is better, though, in practice, cosine with restarts is often used as the default scheduler among machine learning engineers and data scientists like on the machine learning competition website Kaggle. Likewise, one scheduler might be more beneficial to use for one task and one dataset, but might prove inappropriate or unsuited for another combination. The cosine scheduler's periodic nature with hard restarts allows the model to have passes at each mini-batch of the data with different learning rates allowing the model to understand the dataset at different levels compared with the monotonically decreasing polynomial and linear schedulers.

6 Summary and Conclusion

With the advent of large neural networks, robust stochastic gradient descent algorithms (optimizers) and learning rate schedulers are required to reduce training costs and increase model architecture iteration for development. By performing a rigorous survey of different neural network tasks, tabular binary classification, named entity recognition, and image

classification, with various optimizers (`AdamW`, `rmsprop`, and `AdaGrad`) and schedulers (linear decay, polynomial decay, and cosine with restarts), the best combination can be chosen to work reliably regardless of dataset and task. Training a custom linear neural network, and fine-tuning the `distilroberta-base` and `MobileViTv2` model on a Credit Default, CONLL2003, and HAM10000 datasets, respectively, the `AdamW` optimizer along with either the linear decay or cosine with restarts optimizer preformed best. In practice, however, the cosine with restarts may be better, though it requires one more hyperparameter variable with the number of cycles.

In the future, more optimizers, schedulers, datasets, and models can be tested to further diagnose this important problem. This report, at it's current state, merely gives an idea of what is the best combination but falls short of painting the full picture of what optimizer and scheduler is best for a given machine learning task. Furthermore, larger model sizes may decrease the disparity in optimizer and scheduler performance, changing the requirement for the "best" optimizer and scheduler combination.

Appendix

The code and data for this project can be in this project's `github` repository.

References

[1] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* (2023).

[2] Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers* (2010), Springer, pp. 177–186.

[3] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[4] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR abs/2010.11929* (2020).

[5] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research 12*, 7 (2011).

[6] Fukushima, K. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics 5*, 4 (1969), 322–333.

[7] Hinton, Geoffrey; Srivastava, N. S. K. Neural networks for machine learning: Lecture 6, 2014.

[8] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Li, J., Sun, A., Han, J., and Li, C. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering 34*, 1 (2022), 50–70.

[10] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[11] Loshchilov, I., and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983* (2016).

[12] Loshchilov, I., and Hutter, F. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101* (2017).

[13] Mehta, S., and Rastegari, M. Separable self-attention for mobile vision transformers. *arXiv preprint arXiv:2206.02680* (2022).

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (2019).

[15] Rawat, W., and Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Computation 29*, 9 (2017), 2352–2449.

[16] Tjong Kim Sang, E. F., and De Meulder, F. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003* (2003), pp. 142–147.

[17] Tschandl, P., Rosendahl, C., and Kittler, H. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data 5*, 1 (2018), 1–9.

[18] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[19] Yeh, I.-C. Default of credit card clients. UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C55S3H.