Aleksander Lempinen

MLOps approach for system performance optimization for machine learning systems

Master's Thesis in Information Technology

June 12, 2024

University of Jyväskylä

Faculty of Information Technology

Author: Aleksander Lempinen

Contact information: aleksander.lempinen@gmail.com

Supervisor: Tommi Mikkonen

Title: MLOps approach for system performance optimization for machine learning systems

Työn nimi: MLOps lähestymistapa järjestelmän suorituskyvyn optimointiin koneoppimisjärjestelmille

Project: Master's Thesis

Study line: Educational Technology

Page count: 61+0

Abstract: There are numerous practical challenges related to development or operation of machine learning systems in real-world scenarios, and the field of MLOps brings DevOps practices from software engineering to machine learning. This thesis investigated whether using early stopping with system metrics leads to more efficient hyperparameter tuning when resource constraints exist. The experiments conducted measured system performance including mean step time, CPU utilization, and memory utilization on 4 datasets and 4 machine learning algorithms with varying hyperparameters such as batch size and learning rate. Findings indicate that increased mean step time and memory utilization with large batch sizes could potentially be leveraged for early stopping.

Keywords: machine learning, MLOps, DevOps, artificial intelligence, AutoML, hyperparameter optimization, performance

Suomenkielinen tiivistelmä: Koneoppimisjärjestelmien kehittämiseen tai käyttöön liittyy lukuisia käytännön haasteita reaalimaailman skenaarioissa, ja MLOps tuo DevOps-käytännöt ohjelmistotekniikasta koneoppimiseen. Tässä opinnäytetyössä tutkittiin, johtaako varhaisen pysäytyksen käyttäminen järjestelmämetriikoiden kanssa tehokkaampaan hyperparametrien optimointiin, kun on olemassa resurssirajoitteita. Eksperimenteissä mitattiin järjestelmän suorituskykyä, mukaan lukien keskimääräinen askelaika, prosessorin käyttöaste ja muistin käyttöaste neljällä datasetillä ja neljällä koneoppimisalgoritmilla, joiden hyperparametrit, kuten eräkoko ja oppimisnopeus, vaihtelivat. Tulokset osoittavat, että suurten eräkokojen myötä lisääntynyttä keskimääräistä askelaikaa ja muistin käyttöastetta voitaisiin mahdollisesti hyödyntää varhaisessa pysäytyksessä.

Avainsanat: koneoppiminen, MLOps, DevOps, tekoäly, AutoML, hyperparametrioptimointi, suorituskyky

List of Figures

List of Tables

Contents

1 Introduction

Machine learning and artificial intelligence have been a hot topic of discussion in the past decade. While there is a mountain of academic research on machine learning methods and tools, a lack of attention is paid to practical, real-world challenges encountered when developing or operating machine learning systems. DevOps has previously addressed similar challenges in software engineering, and a field of machine learning operations or MLOps, which is DevOps applied to ML, has emerged. MLOps focuses on solving challenges related to operating real-world machine learning systems (Kreuzberger, Kühl, and Hirschl 2023).

Real-world machine learning systems are widely deployed in production in various domains (Cabrera et al. 2023). Examples of machine learning systems in different fields include recommender systems (Li et al. 2023), targeted ads (Domingos 2012), drug design (Domingos 2012), or search engines (Domingos 2012).

Most recent breakthroughs that have generated media attention have been in the fields of computer vision in the form of latent diffusion models (Rombach et al. 2022) such as Stable Diffusion (Stability AI 2022) for generating images from prompts and natural language processing in the form of large language models (Touvron et al. 2023) such as ChatGPT (OpenAI 2022). There have also been great developments in tooling for machine learning, such as Tensorflow (Abadi et al. 2016), Pytorch (Paszke et al. 2019) or scikit-learn (Pedregosa et al. 2011) for model development, Ray (Liaw et al. 2018), Horovod (Sergeev and Del Balso 2018) or DeepSpeed (Rasley et al. 2020) for distributed training and MLFlow (Chen et al. 2020) or Tensorboard (Abadi et al. 2016) for machine learning monitoring.

Despite wide adoption and many successes, there are still challenges with machine learning systems in practice (Dai and Meng 2023). The required amount of computation for machine learning has been on the rise (Sarker 2021), particularly the amount of incoming data has required new solutions such as distributed or federated learning (Dai and Meng 2023). Realistic computational budgets and practical efficiency in real-world scenarios have only recently been started to be researched (Prabhu et al. 2023). According to an OpenAI technical blog, the trend is exponential, and more compute leads to better performance (Amodei and Hernandez 2018). Increased compute requirements also mean increased costs, such as financial, operational, and environmental costs. Strubell et al. (2020) bring attention to the environmental impact of training models and, in particular, hyperparameter tuning, during which costs of training many relatively inexpensive models quickly add up.

In addition to cost, there are other requirements. For example, edge machine learning systems encounter system requirements such as latency and energy use and have limited resources such as memory or compute (Chen and Ran 2019). Ways of meeting these requirements include hyperparameter tuning, reducing the number of parameters in the model, and model compression such as knowledge distillation (Chen and Ran 2019).

Early stopping has been used as a cost optimization technique to reduce training time by stopping training when the performance of the model stops improving on the validation set (Prechelt 1998). More recent work on larger models shows that models might still improve later if training continues for a longer time (Hoffer, Hubara, and Soudry 2018). Using early stopping with other performance metrics, such as system metrics, has yet to be as thoroughly studied.

This thesis aims to investigate whether using early stopping with system metrics leads to more efficient hyperparameter tuning when there are resource constraints. The investigation is limited to a small set of widely available machine learning algorithms and datasets to reduce compute costs. The thesis's theoretical significance is that hyperparameter optimization techniques can be used with system metrics. The practical outcomes are reducing costs and tuning models to fit system performance constraints.

This thesis is structured in the following manner: Chapter 2 contains background information about machine learning, DevOps, and MLOps and how they relate. Chapter 3 describes the performed experiments and their methods and design, including research questions, datasets, and algorithms used, and concludes with the results of the experiments. Chapter 4 revisits the research questions and discusses the interpretation of the results, limitations, related work, and future work. Chapter 5 concludes the thesis by summarizing key findings.

2 The intersection of Machine Learning and Operations

Software involving machine learning adds additional complexity to the overall system. Developing, deploying, and monitoring machine learning systems involves both traditional software system concepts and some new machine learning specific concepts. Section 2.1 introduces machine learning and evaluating model performance from a practical perspective. Section 2.2 introduces DevOps and performance evaluation. Section 2.3 combines machine learning and DevOps for production machine learning systems and introduces hyperparameter optimization and performance prediction.

2.1 Fundamentals of Machine Learning

Real-world applications of machine learning are often messy, with numerous decisions for the developer that can result in different behavior of the machine learning model. This section introduces machine learning from a practical standpoint, including necessary performance metrics for model training and empirical performance evaluation.

2.1.1 Practical machine learning

Writing programs and developing algorithms to complete specific tasks is a labor-intensive task requiring professional programming expertise. A different approach is to develop generic algorithms that can change behavior by learning. The field studying these types of algorithms is called machine learning. Machine learning algorithms learn by applying an optimization algorithm to adjust the set of parameters called a model, and this process is called training the model (LeCun, Bengio, and Hinton 2015).

Machine learning is widely used in applications like search, drug design, or ad placement and can also be known as data mining or predictive analytics (Domingos 2012). Developing machine learning systems, which are systems that are based on machine learning, can be a difficult task. Unlike traditional software development, experiments with both code and data as inputs are central to machine learning development (Zaharia et al. 2018), and reproducibility of the experiments is often problematic. While plenty of research focuses on more efficient machine learning methods, datasets, and data quality, the biggest bottleneck is human cycles (Domingos 2012). Faster development iterations improve the developer experience for machine learning system developers or researchers, and an important metric to pay attention to and optimize for is the mean iteration cycle.

Machine learning can be practiced with two different goals in mind. First is explanatory modeling with the purpose of scientific theory building and testing, and the second is predictive modeling, mainly used outside scientific research (Shmueli 2010). One practical difference is that, unlike predictive modeling, explanatory modeling rarely uses holdout test sets or cross-validation for evaluation (Shmueli 2010). The lack or presence of evaluation on a test set can be used as a heuristic to quickly determine whether a machine learning project is explanatory or predictive. However, even explanatory modeling benefits from evaluating the predictive power (Shmueli 2010). In their paper, Domingos (2012) assume all machine learning is predictive and state that machine learning should generalize beyond the training set. It is important to consider the end goals of a machine learning project because common practices in a research setting might not apply to creating machine learning systems in a practical setting.

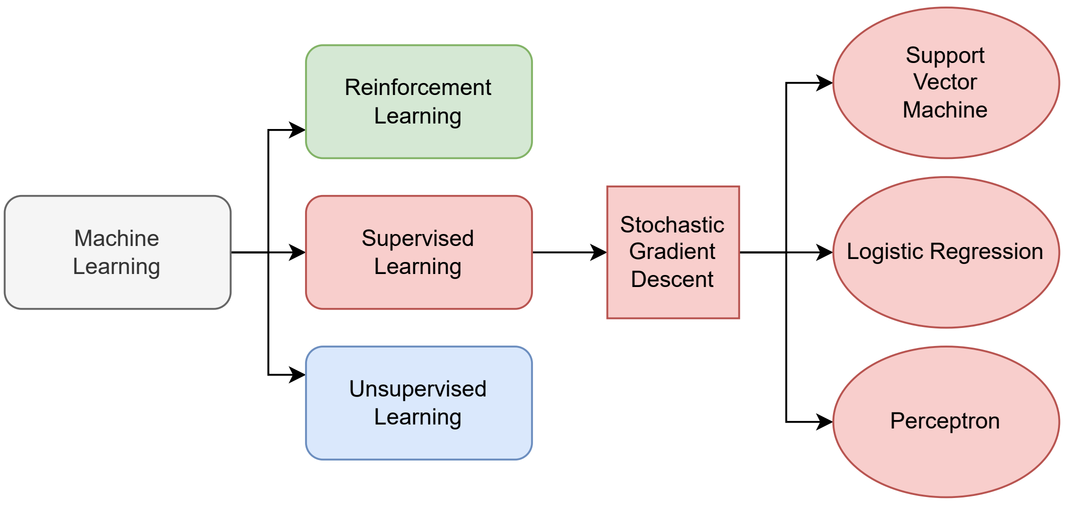

Figure 1. Relation of machine learning algorithms to types of machine learning

Machine learning algorithms can be categorized as supervised, unsupervised, semi-supervised, or reinforcement learning (Sarker 2021). These types of machine learning and some common algorithms are shown in figure 1. The main differences between the types of algorithms are related to whether the model learns using labeled, unlabeled data or by interacting with the environment (Sarker 2021). Unsupervised learning does not require labeled data, an advantage for problems where labels are uncommon (Le et al. 2012). Supervised machine learning can be further split into classification for discrete and regression for continuous labels.

Some machine learning models, such as neural networks, support vector machines, or logistic regression, can be trained in an iterative manner using optimization techniques such as Stochastic Gradient Descent (SGD). Updating the gradient by the learning rate η is a step and is usually performed on a small subset of the data called a batch (Shallue et al. 2019). The magnitude of the update The number of passes over the entire dataset is called epochs (Shallue et al. 2019).

2.1.2 Model evaluation

Performance evaluation of machine learning models is usually done empirically using cross-validation (Forman and Scholz 2009; Sokolova and Lapalme 2009). Cross-validation involves splitting the data into k-folds and using all but one of the folds for training and the last one for validating the performance of the model, after which the procedure is repeated k times with each fold being used for validation (Cawley and Talbot 2010). For example, 3-fold validation would use a third of the data for validation and two-thirds for training repeated three times. The performance metrics collected during the computationally expensive cross-validation are typically averaged (Cawley and Talbot 2010). These types of global averages might not be desirable, and instead of random folds, the data can be sliced according to some criterion, such as by country, and allow detecting performance differences between slices (Breck et al. 2017).

Machine learning training involves minimizing optimization criteria such as log loss, squared hinge loss, or Cauchy-Schwarz Divergence (Janocha and Czarnecki 2017). Depending on the application, different loss metrics are chosen, such as resistance to noisy data or labels (Janocha and Czarnecki 2017). The loss metric is sometimes not informative of model performance, such as with classification tasks. In these cases, performance metrics such as accuracy, precision, recall, specificity, error rate, AUC, and F-score are used (Sokolova and Lapalme 2009; Forman and Scholz 2009). Metrics such as accuracy are well defined, but the final F-score from cross-validation may be computed in several ways, resulting in different results (Forman and Scholz 2009).

Even more informative metrics can be created for specific applications. For example, Torrabla and Efros (2011) developed performance metrics to compare different datasets and determine a "market value" for the data by using the generalization performance of machine learning models on the datasets. Defining the correctness of the prediction is an important part when defining performance metrics (Lin et al. 2014)

2.2 DevOps: principles of Software Development and Operations

DevOps is a well-known topic in the field of software engineering that brings together development, operations, and sometimes quality assurance. This intersection between concepts is demonstrated in figure 2. This section briefly introduces DevOps and provides an overview of the main benefits of continuous integration, deployment, and performance evaluation. Later, it describes the importance of performance metrics with examples and wraps up the section by introducing performance prediction.

2.2.1 Benefits of DevOps

DevOps can be defined as a development methodology bringing development and operations together with a focus on software quality, collaboration between development and operations, process speed and rapid feedback (Jabbari et al. 2016; Mishra and Otaiwi 2020; Waller, Ehmke, and Hasselbring 2015; Perera, Silva, and Perera 2017). Defining DevOps precisely can be difficult as there is no consensus on the exact definition (Smeds, Nybom, and Porres 2015; Jabbari et al. 2016; Mishra and Otaiwi 2020). DevOps can be viewed from different points of view, such as culture, collaboration, automation, measurements, and monitoring (Mishra and Otaiwi 2020; Waller, Ehmke, and Hasselbring 2015). DevOps focuses on speed and quality with incremental changes that are recurrent and continuous (Mishra and Otaiwi 2020). The goal is to bridge the gap between development and operations (Smeds,

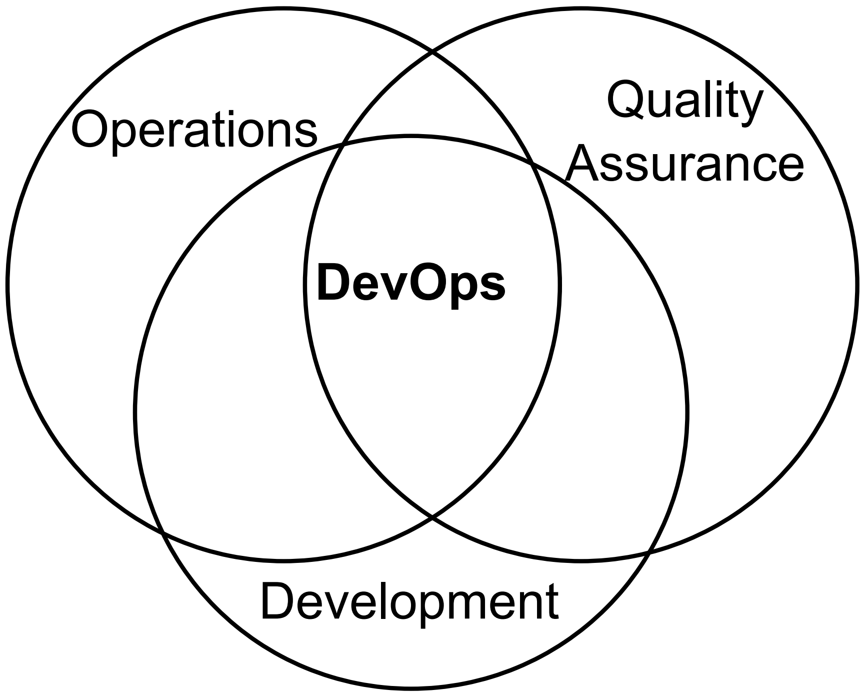

Figure 2. DevOps is the intersection between Development, Operations and Quality Assurance

Nybom, and Porres 2015). This is done through sharing tasks and responsibilities from development to deployment and support (Mishra and Otaiwi 2020).

Continuous integration, continuous deployment, and continuous monitoring are well-known practices in DevOps (Waller, Ehmke, and Hasselbring 2015) describing the automatic nature of integrating, deploying, and monitoring code changes. Feedback includes performance metrics data, which is then fed as input during planning and development (Smeds, Nybom, and Porres 2015). Performance profiling and monitoring are similar activities, and the main difference is whether it is done during the development process or operations respectively (Waller, Ehmke, and Hasselbring 2015) with DevOps bridging the gap between them (Brunnert et al. 2015). Continuous benchmarking allows for detecting performance regressions during continuous integration (Waller, Ehmke, and Hasselbring 2015) and infrastructure monitoring with a feedback loop allows for performance optimization in production (Smeds, Nybom, and Porres 2015).

Performance evaluation is a valuable tool for optimizing the overall system design and tailoring for a specific production environment in addition to correctly sizing resources (Brunnert et al. 2015; Waller, Ehmke, and Hasselbring 2015). Resource demands might change depending on the inputs (Brunnert et al. 2015), making it essential to systematically measure performance based on code changes, configuration changes, and data changes. Performance evaluation is directly tied to defining and collecting performance metrics and monitoring.

2.2.2 Performance evaluation

Performance metrics are fundamental to all performance evaluation activities, such as profiling or monitoring (Brunnert et al. 2015). Common metrics involve measuring the CPU, but other metrics such as memory usage, network traffic, or I/O usage do not have precise definitions (Brunnert et al. 2015). Collecting metrics happens through hardware-based or software monitors instrumented into software through code modification or indirectly, for example, through middleware interception (Brunnert et al. 2015). Metrics can be event-driven, triggering a monitor with every occurrence or based on sampling at fixed time intervals (Brunnert et al. 2015). The types of metrics collected and what information is expected depending on the performance goals and the life cycle of the software (Brunnert et al. 2015).

Metrics can be divided into application metrics such as response time or throughput and resource utilization metrics such as CPU utilization or available memory (Brunnert et al. 2015). Little peer-reviewed research is available with specifics on which metrics are to be collected or how they are defined. Kounev et al. (2020) in their textbook on systems benchmarking, bring up the following quality attributes for benchmark metrics: easy to measure, repeatable, reliable, linear, consistent, and independent. Most metrics will not satisfy all the above quality attributes, and aggregated higher-level composite metrics are required (Kounev, Lange, and Von Kistowski 2020). Cloud computing has introduced more objectives to optimize and metrics for monitoring such as energy consumption, privacy, or time to scalability (Aslanpour, Gill, and Toosi 2020).

Measurement-based performance evaluation requires a system to test while model-based performance evaluation allows predicting the performance of the future system (Brunnert et al. 2015). This type of performance prediction allows for better planning and comparing use cases, primarily when an existing legacy system exists with measured performance metrics (Brunnert et al. 2015).

2.3 MLOps: bridging the gap between Machine Learning and DevOps

MLOps is a new concept for building and running real-world machine learning systems. MLOps can be described as the intersection between machine learning and DevOps, as demonstrated in figure 3. This section introduces the concept of MLOps and provides context for the types of problems it aims to solve. Later in the section, the concepts of hyperparameter optimization, performance prediction, and early stopping are introduced. The section finishes with performance metrics related to machine learning systems, their business objectives, and overall system performance.

2.3.1 Production machine learning systems

MLOps bridges the gap between ML practitioners and DevOps (Moreschi et al. 2023). While machine learning research focuses on improving models, the industry needs to be able to design production-ready machine learning pipelines (Posoldova 2020). The data often used for research is of higher quality than real-world data that is often messy, unstructured, and unlabeled (Posoldova 2020). Continuous integration, deployment, and automated testing are also relevant to machine learning systems (Posoldova 2020), which are familiar concepts from DevOps. A new concept of MLOps addresses this issue of designing and maintaining machine learning systems just like DevOps addressed it for traditional software (Kreuzberger, Kühl, and Hirschl 2023).

Managing technical debt is even more critical in machine learning systems because of machine learning specific issues that cannot be solved with traditional methods (Sculley et al. 2015). The main culprit for the challenges with machine learning systems is that data

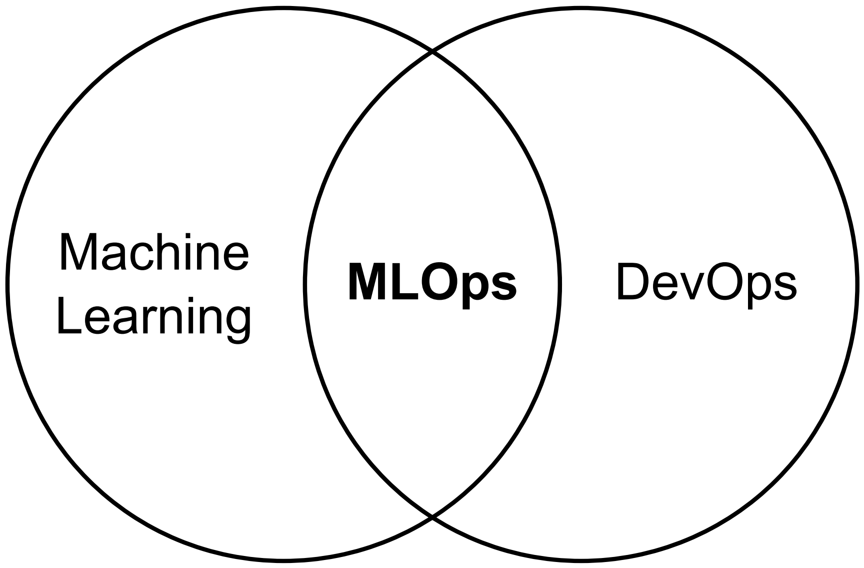

Figure 3. MLOps is the intersection between Machine Learning and DevOps.

changes the system's behavior and cannot be expressed with code alone (Sculley et al. 2015). Challenges like entanglement, correction cascades, or feedback loops are common with machine learning systems and are difficult to diagnose with common tools (Sculley et al. 2015).

Requirements for a machine learning system are different depending on the task. For example, speech and object recognition might have no particular performance requirements during training but have strict latency and computational resource restrictions when deployed to serve large amounts of users (Hinton, Vinyals, and Dean 2015). MLOps has to consider both machine learning performance metrics that are familiar to machine learning and software performance metrics that are familiar to DevOps and software engineering. Feedback from metrics collected during development and monitoring production systems are core MLOps principles (Kreuzberger, Kühl, and Hirschl 2023). For example, possible meta-level requirements include users requesting data deletion and prohibitions on specific features like age or deprecated sources (Breck et al. 2017).

Performance measuring software is not new, but ML brings additional challenges in the form of models and data, which requires a modified approach (Breck et al. 2017). It is also important to note that not every data scientist or machine learning engineer working on machine learning systems has a software engineering background (Finzer 2013) and might lack the necessary knowledge to apply software engineering best practices to machine learning systems. Machine learning system monitoring must be carefully designed (Sculley et al. 2015). Hyperparameter optimization is a kind of performance optimization that aims to improve machine learning metrics. Training the model to completion is not always necessary to verify that the training code is correct, and the training loss is decreasing (Breck et al. 2017).

2.3.2 Hyperparameter optimization

Parameters given as part of a configuration to the machine learning model are called hyperparameters (Yang and Shami 2020). Examples of hyperparameters include learning rate, number of layers in a neural network, regularization coefficients, batch size, step size, or initialization conditions (Maclaurin, Duvenaud, and Adams 2015; Baker et al. 2017; Breck et al. 2017). Hyperparameter tuning or hyperparameter optimization can be defined as finding the optimal hyperparameter values by searching through possible hyperparameter values (Baker et al. 2017). This hyperparameter search can also demonstrate whether the training is stable and reliable (Breck et al. 2017).

The main goal of hyperparameter optimization is to reduce the amount of expert labor required for creating high-performance machine learning models (Baker et al. 2017). Another benefit of finding optimal hyperparameters is that it can help achieve state-of-the-art performance in machine learning systems (Maclaurin, Duvenaud, and Adams 2015). Hyperparameter optimization techniques include grid search, random search, gradient-based optimization, and Bayesian optimization, and they have different benefits and limitations (Yang and Shami 2020).

Similar concepts to hyperparameter optimization are neural architecture optimization and meta modeling where model structure or modeling algorithm is treated as a tunable parameter (Baker et al. 2017). This allows for automating the creation of neural networks from scratch (Baker et al. 2017). The amount of potential neural network architecture configurations is large, and checking them is computationally expensive (Baker et al. 2017).

Tuning hyperparameters is generally a difficult task (Maclaurin, Duvenaud, and Adams 2015). Traditional hyperparameter tuning methods such as Bayesian optimization are unfeasible for more than 10-20 hyperparameters (Maclaurin, Duvenaud, and Adams 2015). More advanced techniques are required if a larger amount of tunable hyperparameters is desired. Performance prediction is a crucial step to reduce the computation required for neural architecture search and hyperparameter optimization (Baker et al. 2017). Memory consumption, power consumption, and training time are relevant considerations that can be taken into account by setting boundary conditions to whether the hyperparameter tuning trial is worthy of continuing (Yu and Zhu 2020).

Training models faster can allow for using more data for better model performance and for using more complex models in new types of situations (Shallue et al. 2019). During the training of machine learning models, the main focus is on achieving good model performance and the associated costs (Shallue et al. 2019). Cost can be measured in training time or the price for hardware but is better measured by time or hardware price per training step with compute budgets defined either in the required number of steps or used training time (Shallue et al. 2019). It is essential to efficiently use the available compute budget because training on even simple datasets can require large amounts of computation for each configuration of hyperparameters to saturate model performance (Shallue et al. 2019). It is essential to apply compute resources efficiently in realistic workloads, which is a combination of the dataset, training algorithm, and model (Shallue et al. 2019).

2.3.3 Performance prediction and early stopping

Data gathered at the beginning of model training can be used to predict the performance of the trained model given the chosen hyperparameters (Baker et al. 2017). A small sample of hyperparameter configurations can be used for training a performance prediction model, which then can be used to predict the performance for the rest of hyperparameter configurations with only a small amount of training (Baker et al. 2017).

Early stopping is a technique in which model training is halted before completion to avoid wasting computational resources (Prechelt 1998). Early stopping can be based on a threshold value decided upon ahead of time or based on a performance prediction model (Baker et al. 2017). Low thresholds for rejection of suboptimal solutions will radically reduce the amount of computation required but run the risk of rejecting an optimal solution as well (Baker et al. 2017).

In addition to machine learning performance metrics and system performance metrics, machine learning systems will have their performance metrics tied to product or organization metrics such as user churn rate or click-through rate (Shankar et al. 2022). From a machine learning system performance perspective, important metrics include CPU usage, GPU usage, task completion time, inference time, and latency (Cardoso Silva et al. 2020). Choosing the right metrics to evaluate a machine learning system is essential, and the metrics will differ for different machine learning systems (Shankar et al. 2022).

3 Methods

The approach for the thesis is empirical and experimental, as is common in machine learning and software engineering research. Section 3.1 describes the research methodology used in this thesis and introduces the research questions. Section 3.2 introduces the experimental setup, including software and hardware, datasets, algorithms, metrics, and specific experiment workflow. Section 3.3 contains the details about the experiments and the results of the experiments.

3.1 Methodology

This thesis uses a methodology for machine learning experiment design (Fernandez-Lozano et al. 2016). The methodology, as shown in Figure 4, consists of a workflow with the following steps: Dataset, Data Preprocessing, Model Learning, and Best Model Selection. The main focus of the thesis is on Model Learning and Best Model Selection with an emphasis on using system performance metrics. Advanced preprocessing techniques or achieving state-of-the-art model performance are out of the scope of this thesis.

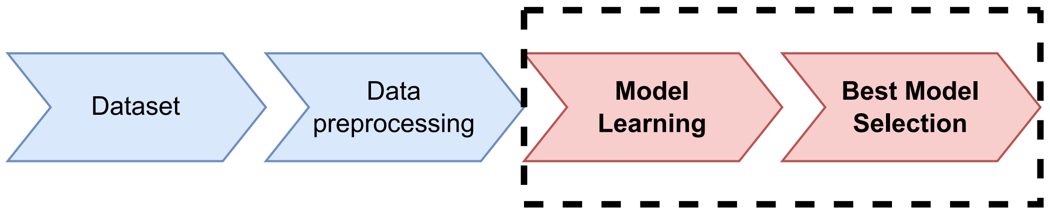

Figure 4. Methodology workflow steps with the focus on model learning and best model selection

This master's thesis asks the following research questions:

- *RQ1*: How does system performance change over time during model training?
- *RQ2*: How do changes in hyperparameters affect system performance during model training?
- *RQ3*: How does early stopping on system performance criteria affect computational budgets during model training?

3.2 Experimental setup

The experimental setup was chosen to be realistic and representative of a machine learning practitioner using common machine learning tools on a development machine. The available computational resources for the thesis limit the scope of investigated workloads. This section first describes the hardware and software used for the experiments. Afterwards, the focus is on workloads, datasets, and algorithms used in the experiments and the metrics to be collected and used for evaluation. Finally, the exact workflows that were used for each experiment are described in detail.

3.2.1 Software and Hardware

Experiments were performed using Ray Tune (2.7.1) (Liaw et al. 2018). MLFlow (2.7.1) (Chen et al. 2020) was used for recording metrics and tracking experiments. Scikit-learn (1.3.2) (Pedregosa et al. 2011) for training, collecting machine learning performance metrics, and evaluating machine learning models. Psutil (Rodola 2023) was used for collecting system performance metrics from the operating system. The hardware used to perform the experiments consisted of Intel Core i7-9700 @ 3.00GHz CPU and Nvidia 3060 GPU.

3.2.2 Datasets

OpenML (Vanschoren et al. 2014) was a source of benchmarking datasets for both classification (Bischl et al. 2017) and regression (Fischer, Feurer, and Bischl 2023) tasks. In total, two classification task and two regression task datasets summarized in table 1 were chosen to keep the amount of computation reasonable.

The *mnist_784* dataset consisted of 70000 images of handwritten digits, with each feature representing a pixel with the task of classifying which digit the image represents. The *diabetes* dataset consisted of 785 measurements of female patients with the task to classify whether the patient tests positive for diabetes. The *wave_energy* dataset consisted of 72000 different positions for 16 buoys with the regression task to predict the total amount of energy produced. The 16 wave energy converter features were dropped as the target variable is total energy. The *red_wine* dataset consisted of 1599 measurements of red wine samples with the regression task of predicting the quality of the wine.

Dataset	Type	Task	Instances	Features
mnist_784	image	classification	70000	785
diabetes	tabular	classification	768	9
wave_energy	tabular	regression	72000	33
red_wine	tabular	regression	1599	12

Table 1. Summary of the datasets used.

3.2.3 Algorithms

Algorithms were chosen to support training in batches without being computationally heavy. Linear regression, perceptron, logistic regression, and support vector machine (SVM) are based on stochastic gradient descent (SGD) implementation found in Scikit-learn (Pedregosa et al. 2011). Algorithms and hyperparameters are summarized in Table 2. Model training, evaluation, and hyperparameter optimization were performed in parallel with each worker process using one CPU core each.

Hyperparameters such as batch size and learning rate were selected using grid search, and the search space was determined with preliminary experiments so that the optimal solution is not too close to the boundaries. Batch size search space was $\{30, 300, 3000, 30000\}$. The learning rate search space was $\{0.1, 0.01, 0.001, 0.0001\}$.

Algorithm	Loss	Hyperparameters
Linear regression	squared	batch size, learning rate
Perceptron	hinge	batch size, learning rate
Logistic regression	log	batch size, learning rate
Support Vector Machine	hinge	batch size, learning rate

Table 2. Summary of the algorithms

3.2.4 Metrics and evaluation

Metrics to be evaluated can be divided into machine learning metrics and system performance metrics and are summarized in Table 3. Machine learning metrics consisted of training loss, validation loss, accuracy for classification, and root mean squared error for regression, respectively. System compute performance was measured through mean training step time and CPU utilization percentage. System memory performance was measured through memory use of the process, and computational budget was measured as elapsed wall-time required for training the model. Training loss was computed with each training step, and the rest of the metrics were computed every 100 training steps. Machine learning metrics were computed using scikit-learn (Pedregosa et al. 2011), and system performance metrics were collected from the operating system using psutil (Rodola 2023).

Metric	Type
training loss	machine learning
validation loss	machine learning
accuracy	machine learning
root mean squared error	machine learning
mean training step time	system performance
total training time	system performance
CPU utilization (%)	system performance
memory (MB)	system performance

Table 3. Summary of the metrics

In accordance with Ray documentation (The Ray Team,) to avoid double counting memory used by the object store, the memory usage of the worker was computed in the following way:

$$\text{memory} = \text{resident set size (RSS)} - \text{shared memory usage (SHR)}$$

Machine learning models were validated by splitting the dataset into a 70% training set and a 30% test set and only using the training set for training the model and only the test set to compute the test loss.

3.2.5 Machine learning experiment workflow

Experiments consisted of dataset loading, preprocessing, and several model training and evaluation runs with different configurations. Classification workloads consisted of MNIST and Diabetes datasets with SVM, perceptron, and, in the case of the Diabetes dataset, logistic regression algorithm. Regression workloads consisted of red wine and wave energy datasets with the SVM algorithm.

Dataset loading and preprocessing consisted of downloading the dataset, splitting into test and train sets and loading them into shared memory. Each algorithm and hyperparameter combination was a separate run using Ray Tune. Model training was performed by first initializing the model and then fitting the model one batch at a time and collecting metrics every 100 steps.

To ensure that measurements are not sensitive to other processes running on the system, the metrics are averaged over three runs for each workload and hyperparameter.

The average of all the runs was visualized for each metric and inspected for clear patterns with the focus on system performance metrics.

3.3 Experiments and Results

The first experiment was performed to determine whether system performance is constant during model training. The second experiment was conducted to determine how changes in the hyperparameters affect system performance metrics. For both experiments, the collected metrics are visualized and interpreted.

3.3.1 Experiment 1: Changes in system performance during model training

Both training loss and validation loss decreased during training classification workloads on the MNIST dataset, as seen in Figure 5. Training and validation loss also decreased on regression workloads on Red Wine and Wave Energy datasets shown in Figure 6.

Accuracy did not increase or decrease during model training on the MNIST dataset but did increase at the beginning of training before leveling off for the diabetes dataset, as demonstrated by Figure 7. There was little difference between different algorithms with the same dataset, but a clear difference between datasets.

Mean step time shown in Figure 8 has dips and peaks during training, but overall is level for each workload. There was a clear difference of mean step time between the MNIST dataset and the rest of the datasets.

CPU utilization Figure 9 remained level for each of the workloads. The CPU utilization was different for most of the workloads.

Memory utilization Figure 10 increased in the beginning training for Wave Energy and MNIST datasets before leveling off. Diabetes and Red Wine memory utilization remained level during training. Memory utilization was higher for the MNIST and Wave Energy datasets than for the Diabetes and Red Wine datasets.

3.3.2 Experiment 2: Effects of hyperparameter changes on system metrics

Test loss for classification and regression workloads depended on the batch size, as seen in Figure 11 and Figure 12. For smaller batch sizes, the test loss curve was less steep and converged slower than for large batch sizes, except with Logistic Regression on the Diabetes dataset. Too large of a learning rate did not lead to convergence, as seen in Figure 13 and Figure 14.

Batch size had mixed effects on classification workloads, as seen in Figure 15. On the MNIST dataset, batch size had no effect on accuracy. On the Diabetes dataset, smaller batch sizes resulted in lower accuracy, especially with Logistic Regression. Learning rate had no effect on accuracy, as seen in Figure 16.

Mean step time was constant with smaller datasets such as Diabetes or Red Wine and depended on the batch size with larger datasets such as MNIST and Wave Energy. As can be seen in Figure 17 and Figure 18 very large batch sizes resulted in higher mean step time.

Learning rate had no effect on mean step time, as seen in Figure 19 and Figure 20.

CPU utilization did not depend on batch size or learning rate for classification workloads as shown in Figure 21 and Figure 23 or for regression workloads as shown in Figure 22 and Figure 24.

Memory utilization was constant with smaller datasets but not with larger datasets. As can be seen in Figure 25, memory utilization was higher for large batch sizes for the MNIST dataset, and in Figure 26, memory utilization was higher for large batch sizes for the Wave Energy dataset. Learning rate had no effect on memory utilization as seen in Figure 27 and Figure 28.

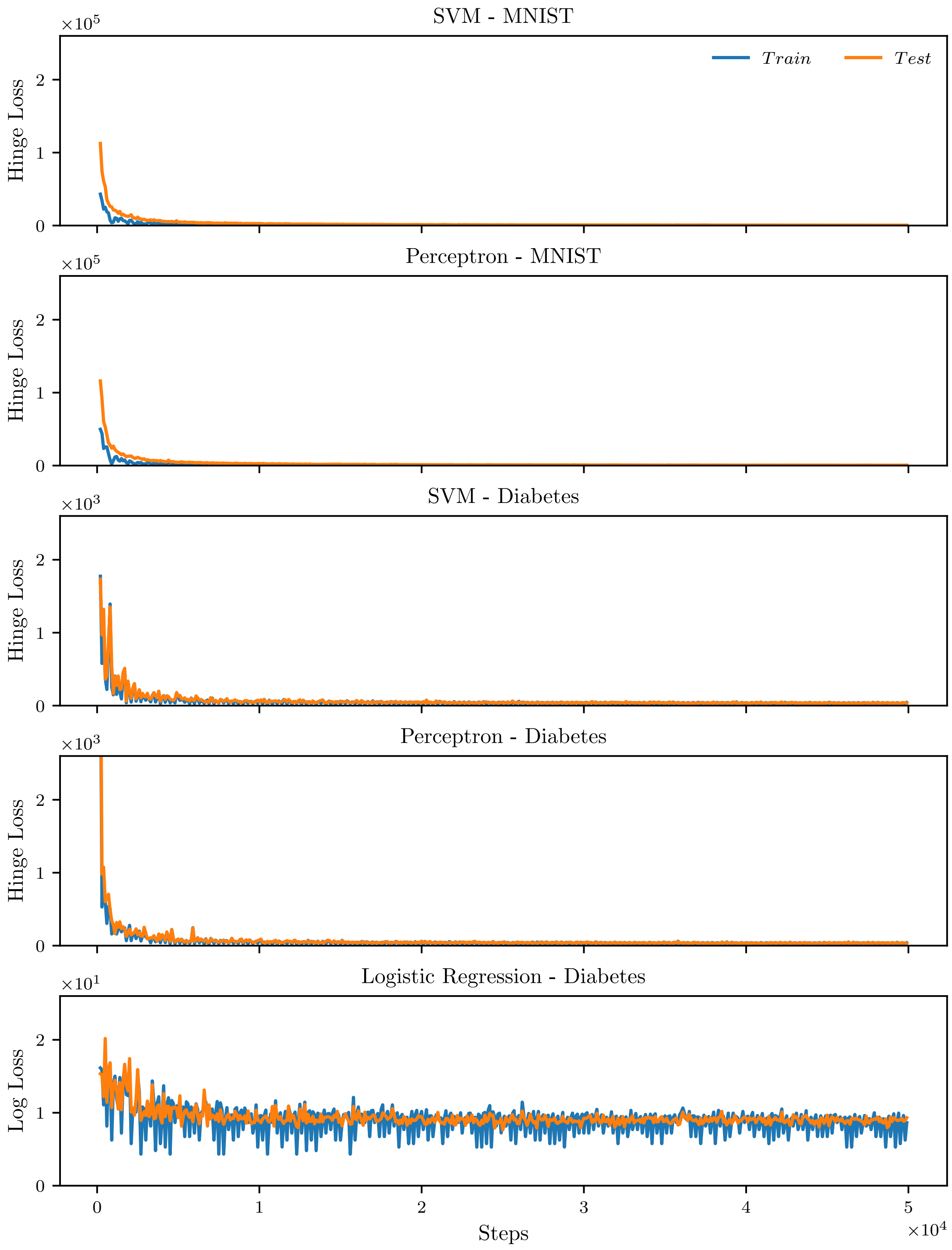

Figure 5. Change in training and test loss during model training with classification workloads

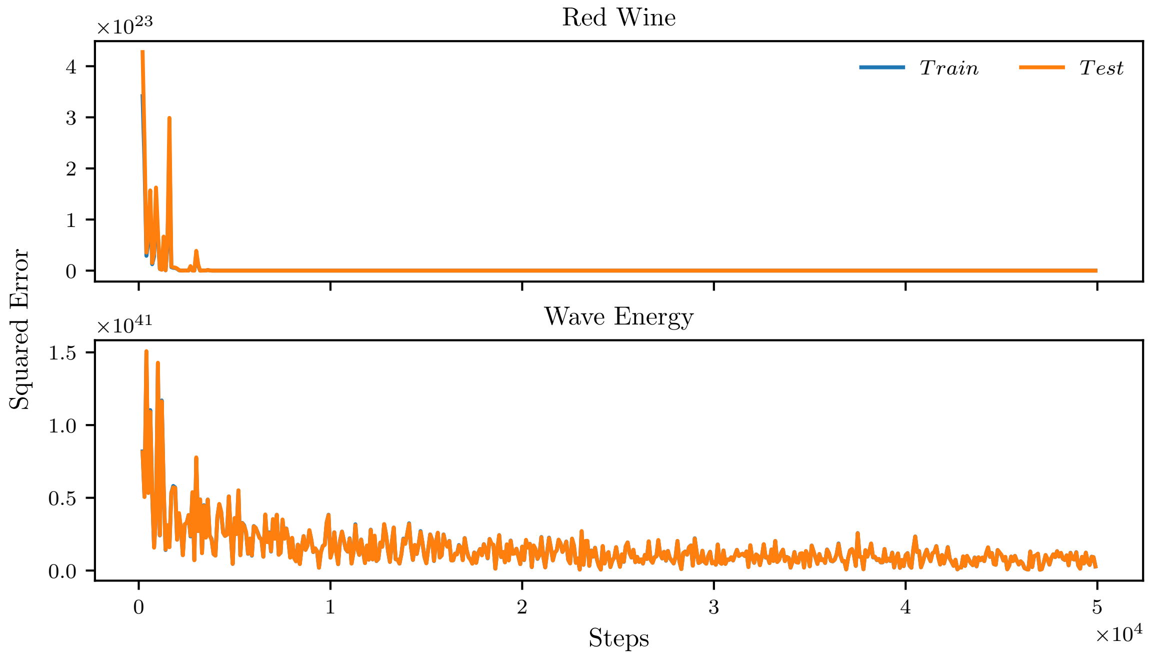

Figure 6. Change in training and test loss during model training with regression workloads

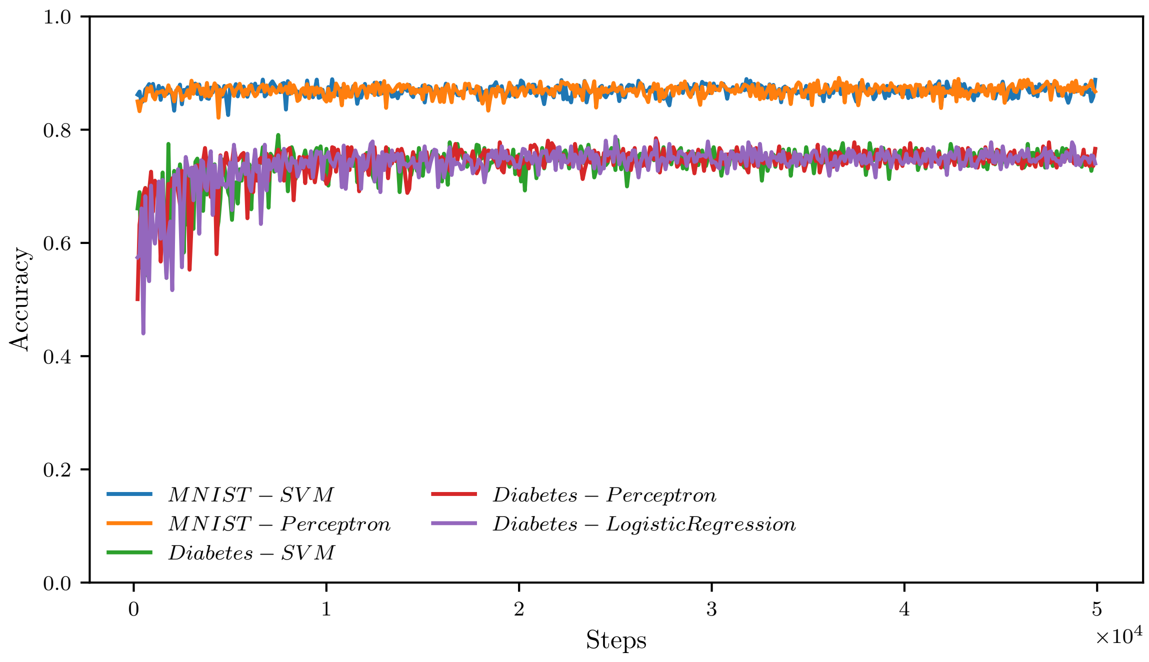

Figure 7. Change in classification accuracy during model training

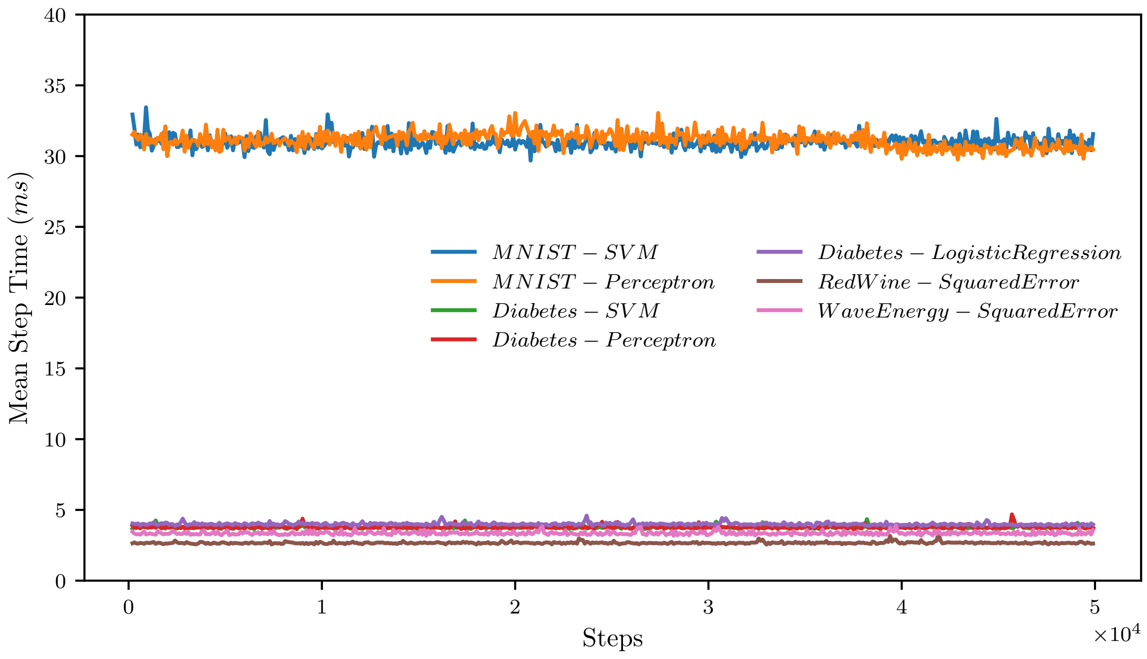

Figure 8. Change in mean step time during model training

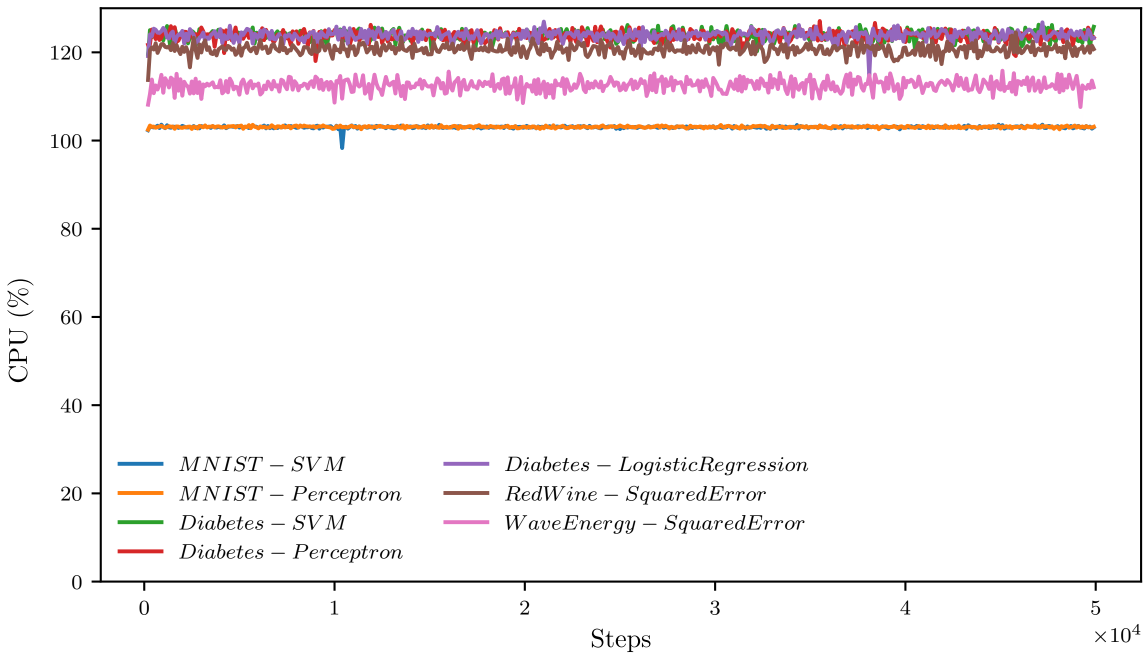

Figure 9. Change in CPU utilization during model training

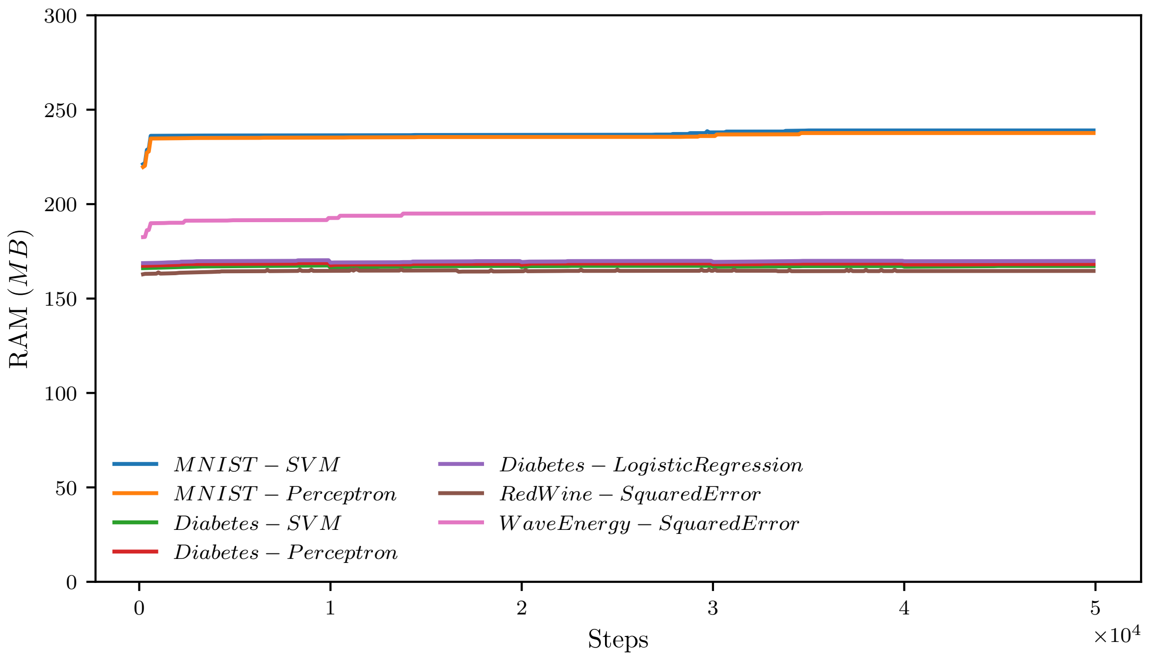

Figure 10. Change in memory utilization during model training

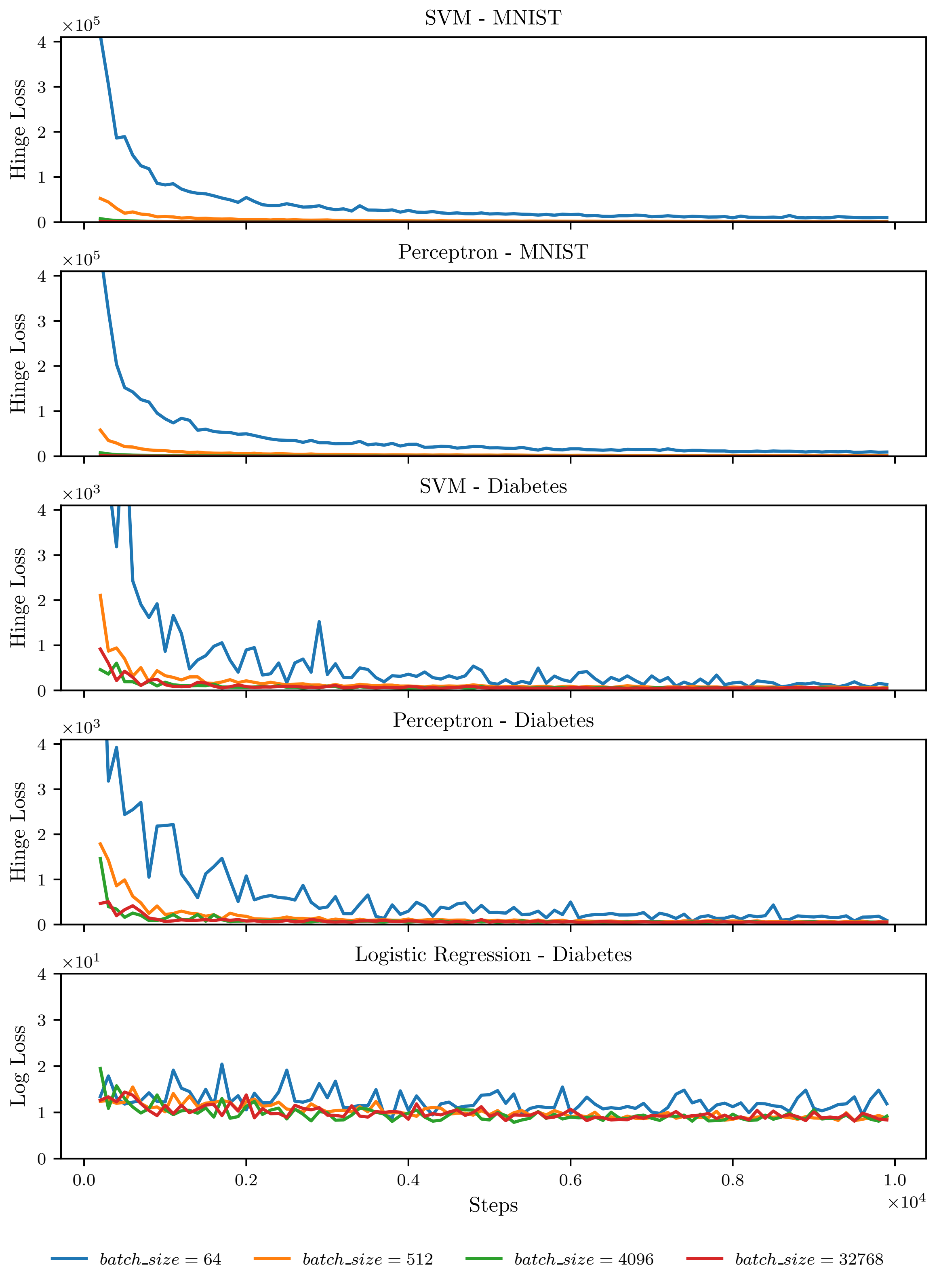

Figure 11. Effects of different batch sizes on test loss on the MNIST and Diabetes datasets

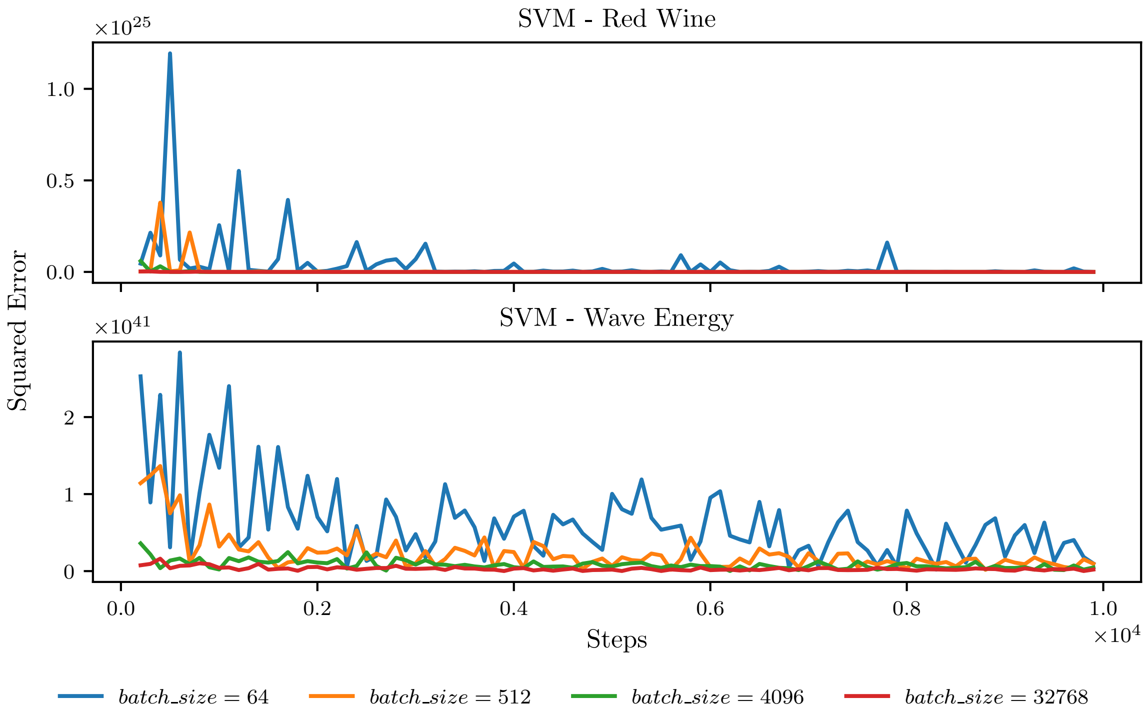

Figure 12. Effects of different batch sizes on test loss on the Red Wine and Wave Energy datasets

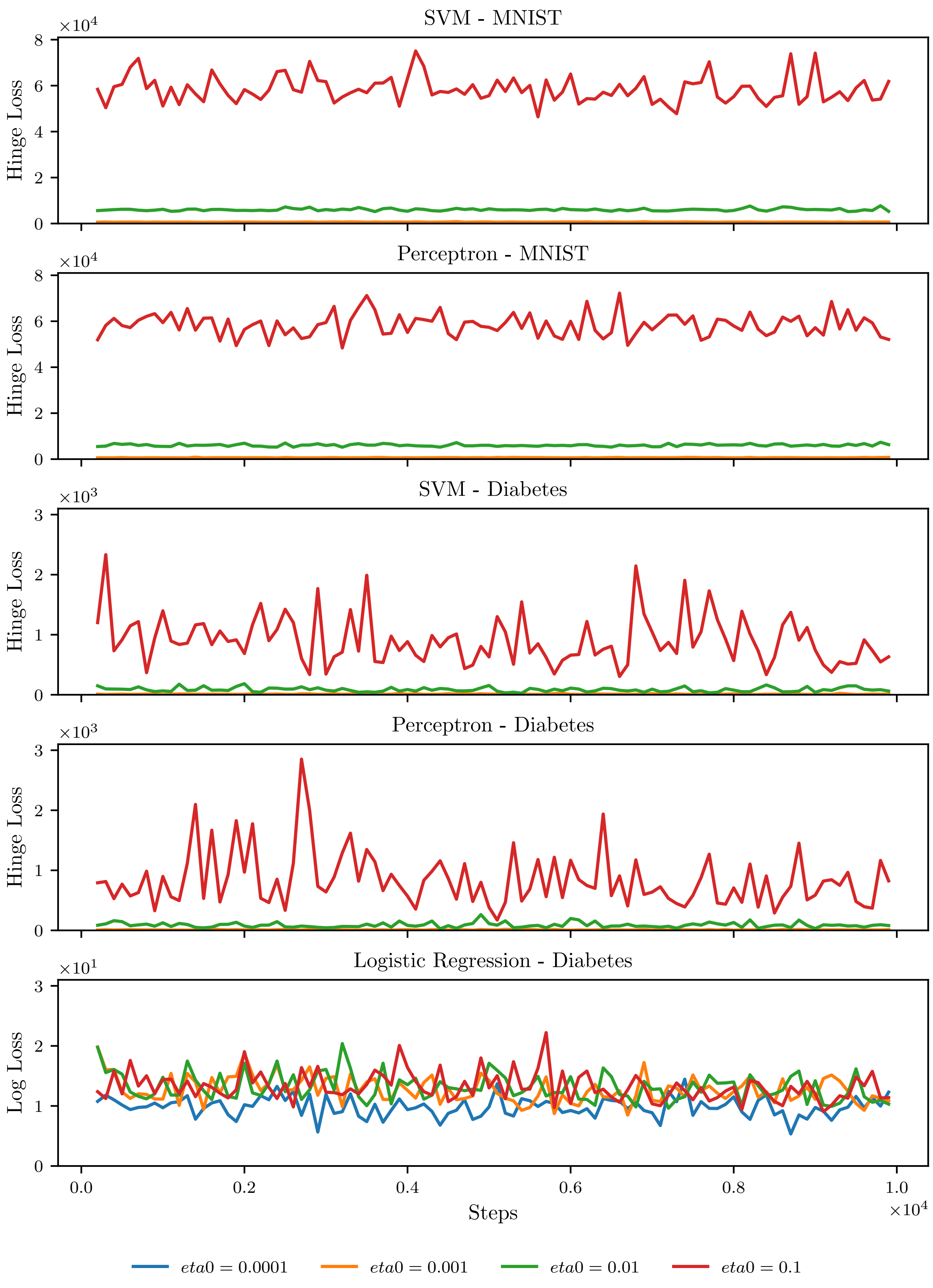

Figure 13. Effects of different learning rates on test loss on the MNIST and Diabetes datasets

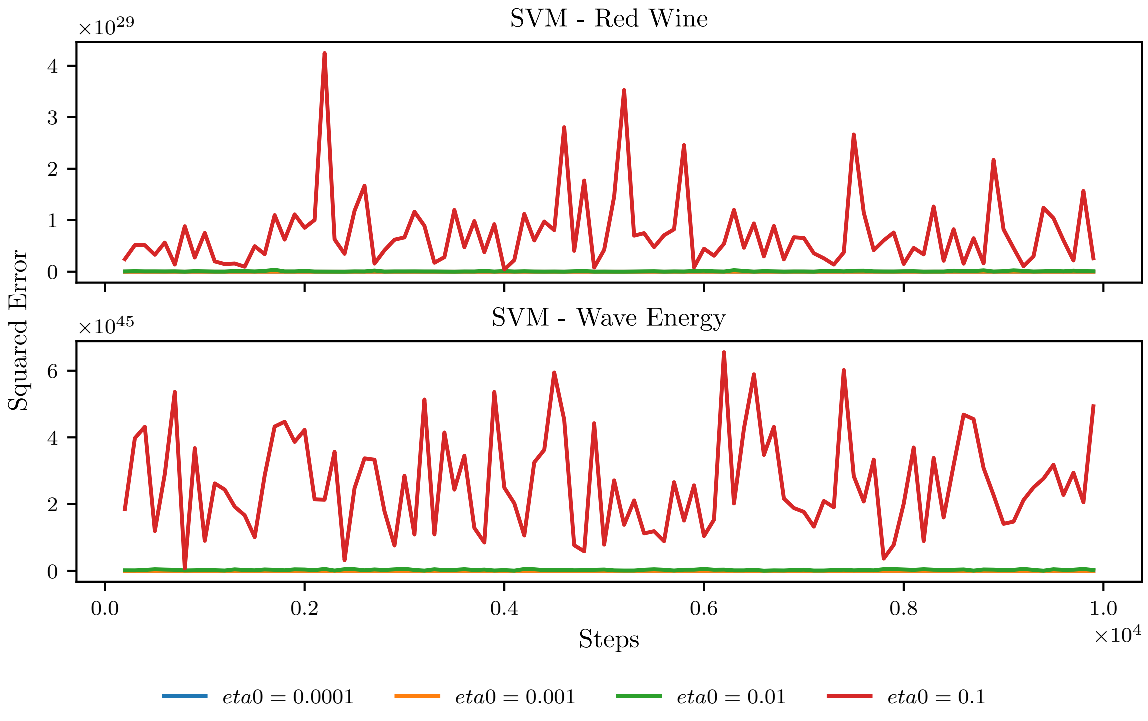

Figure 14. Effects of different learning rates on test loss on the Red Wine and Wave Energy datasets

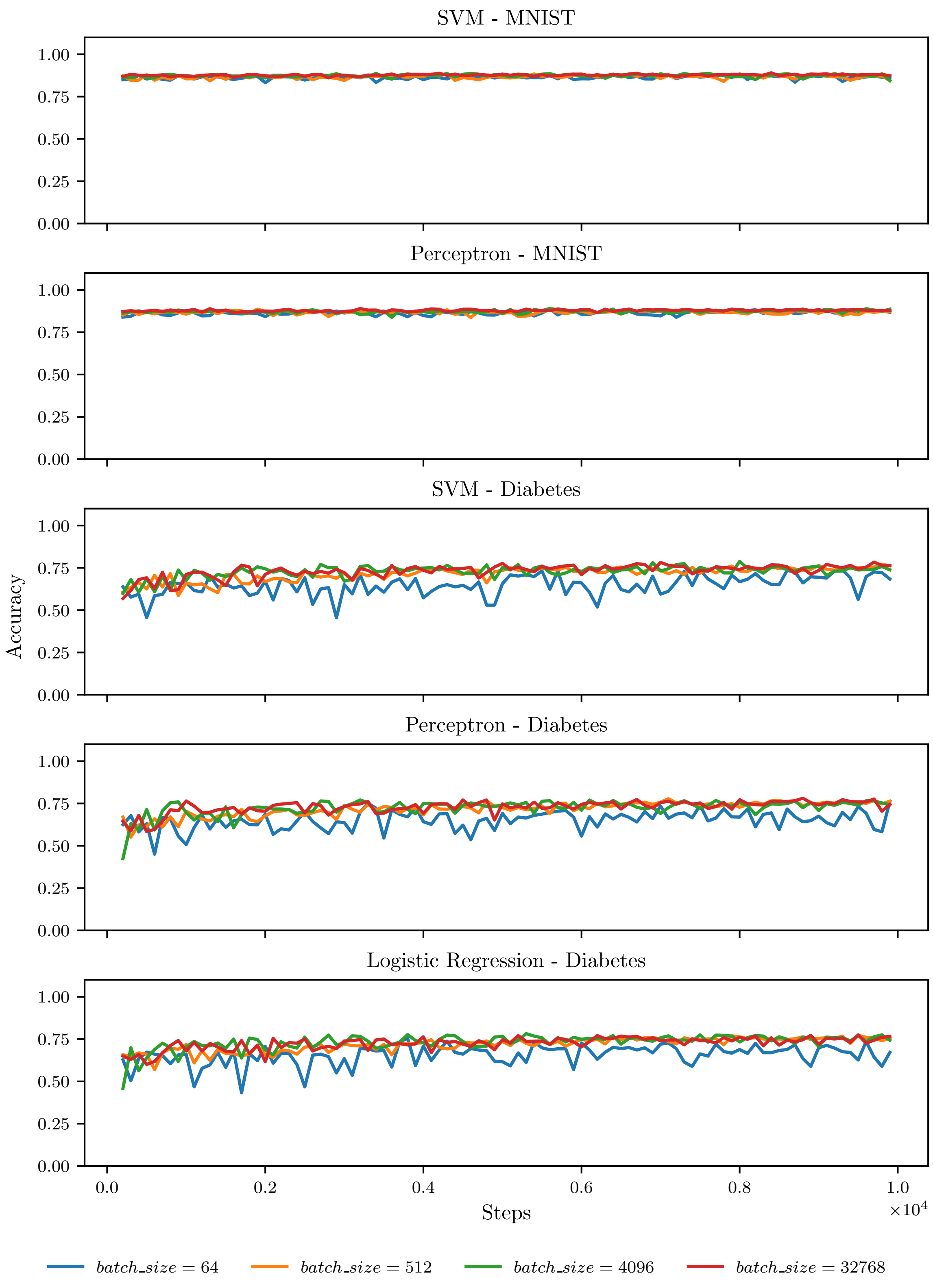

Figure 15. Effect of different batch size on accuracy with classification workloads

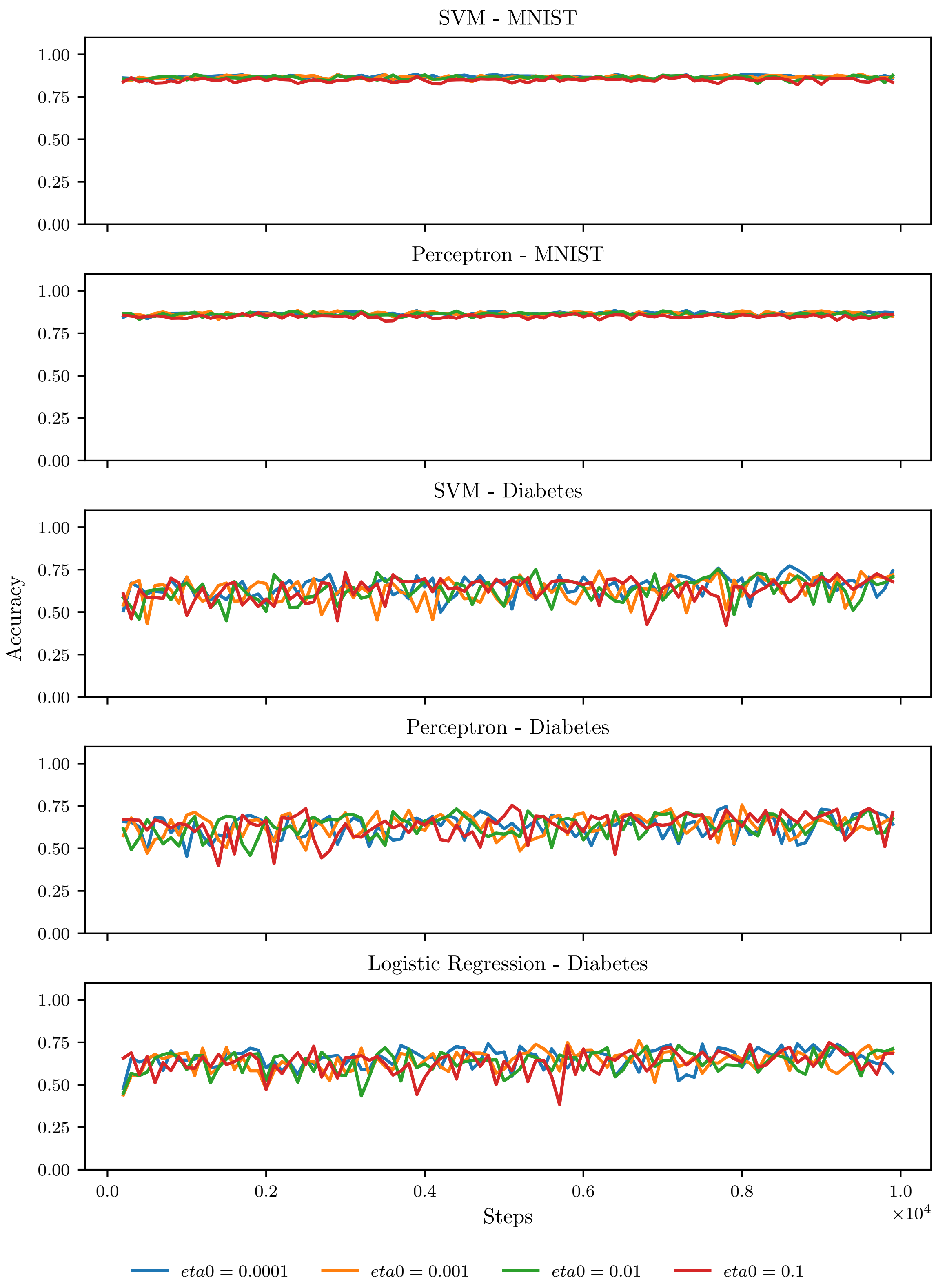

Figure 16. Effect of learning rate on accuracy with classification workloads

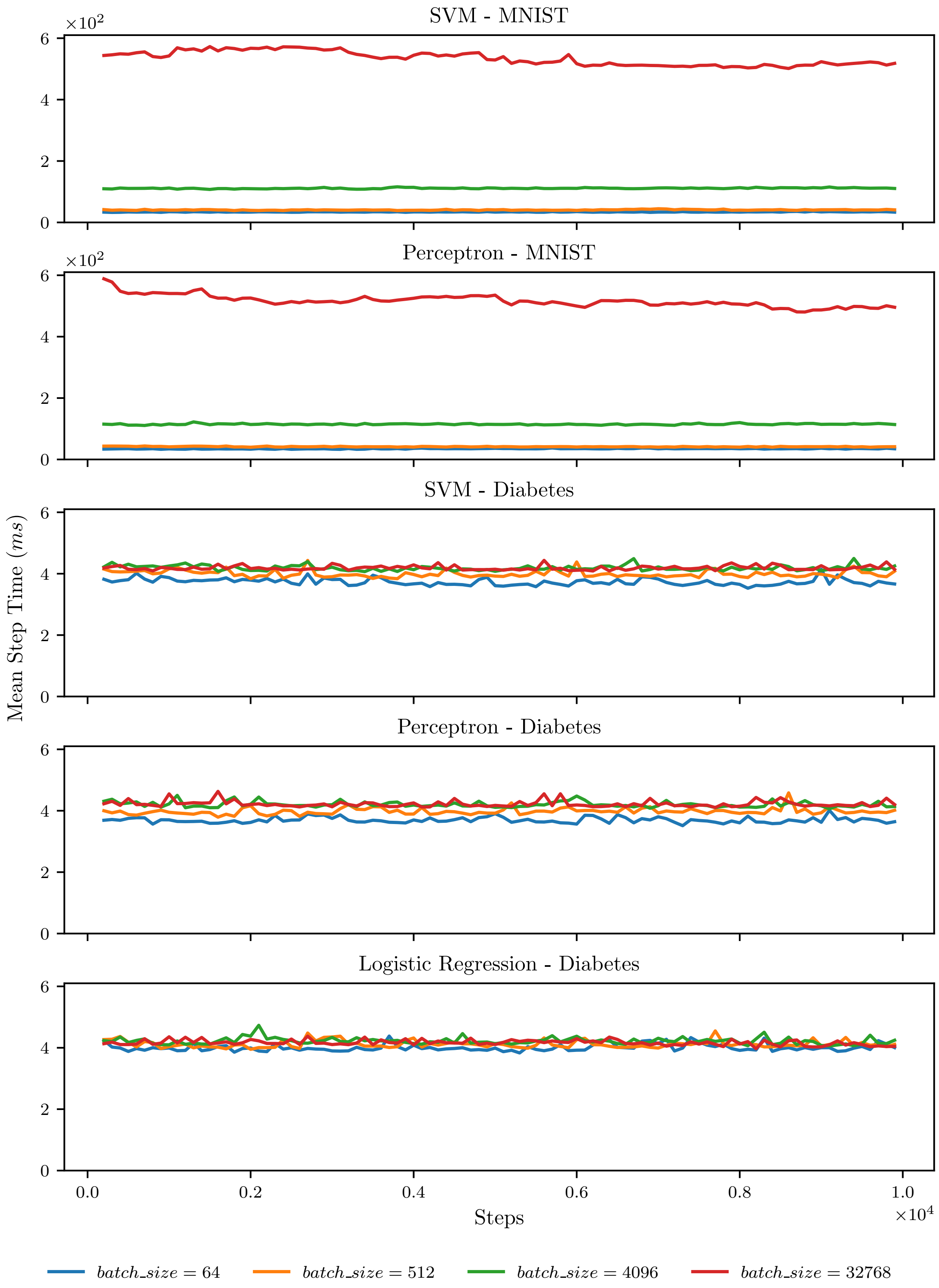

Figure 17. Effect of batch size on mean step time with classification workloads

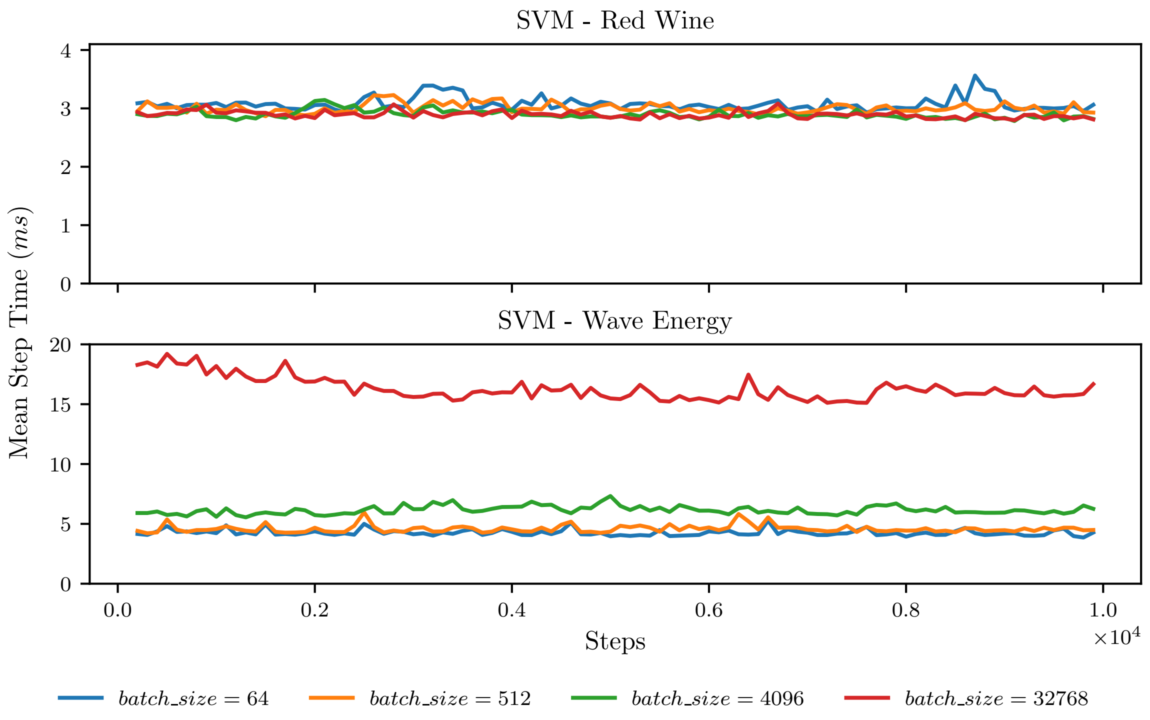

Figure 18. Effect of batch size on mean step time with regression workloads

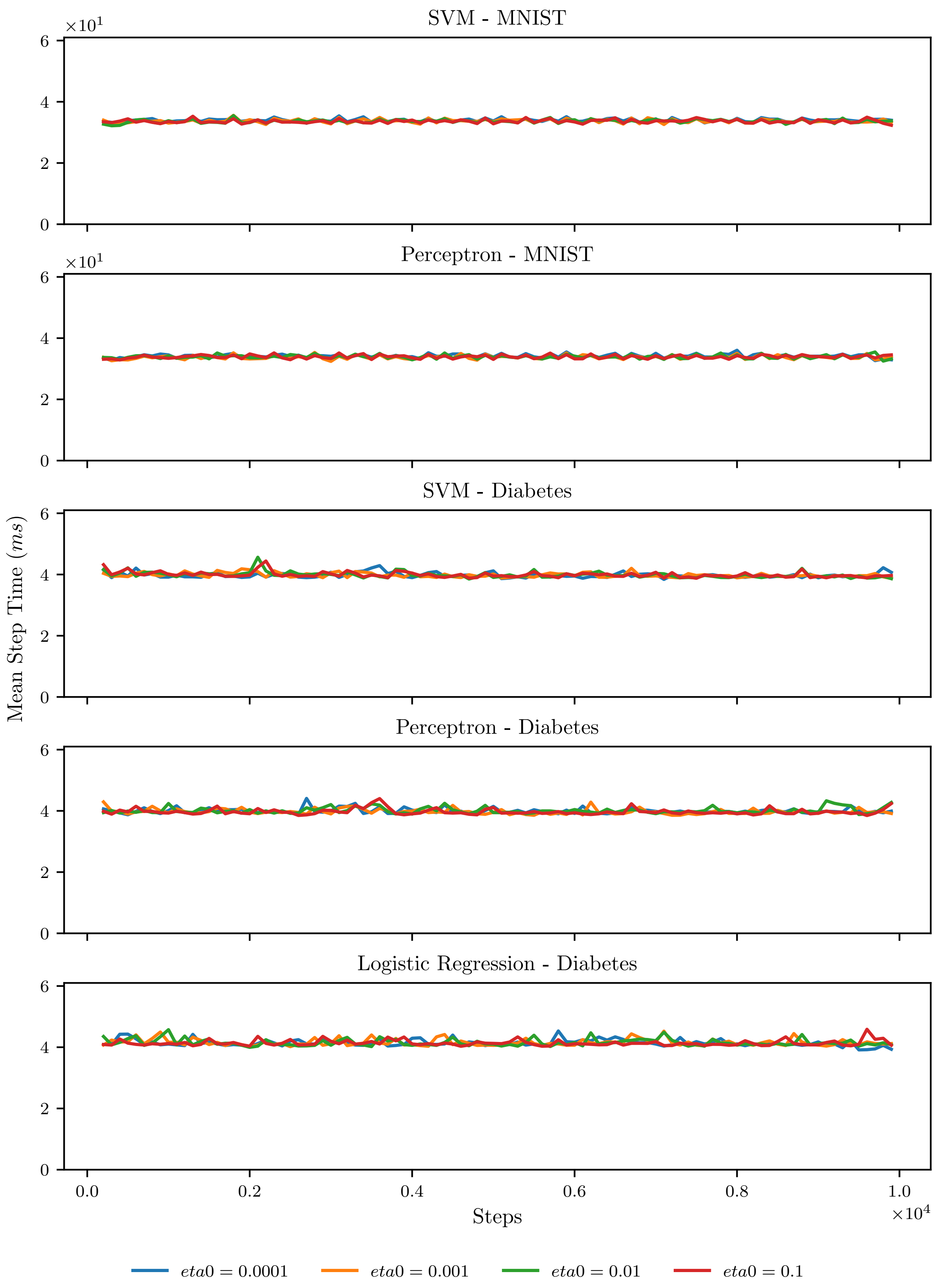

Figure 19. Effect of learning rate on mean step time with classification workloads

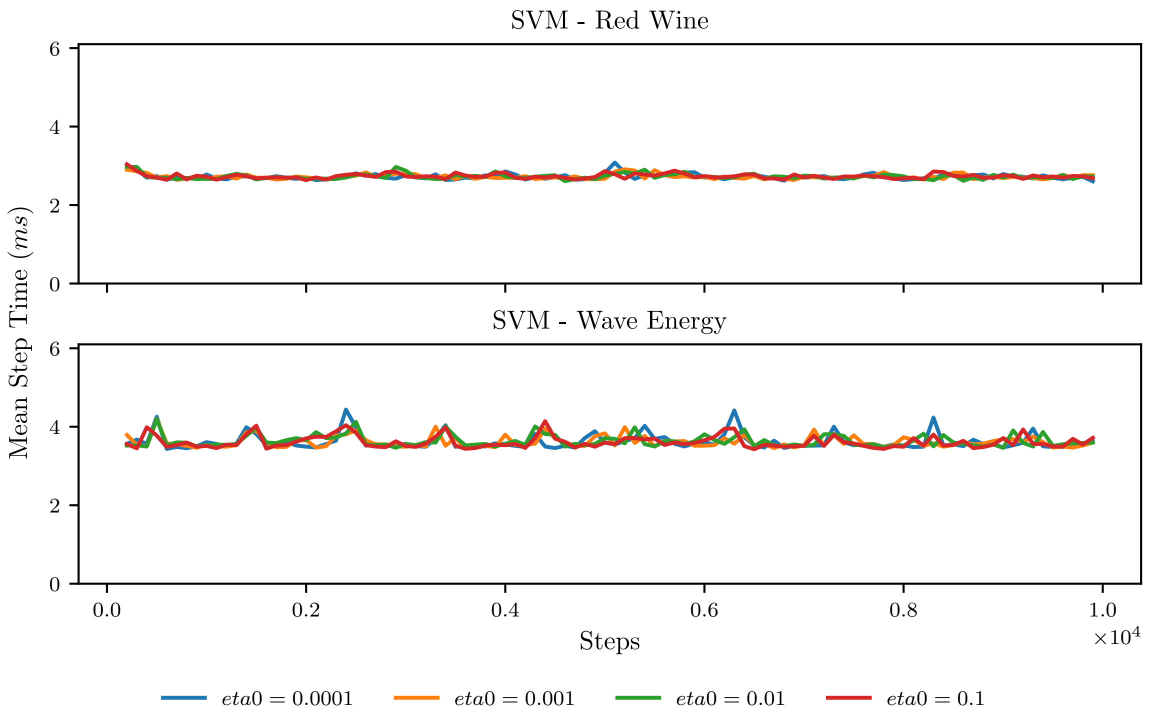

Figure 20. Effect of learning rate on mean step time with regression workloads

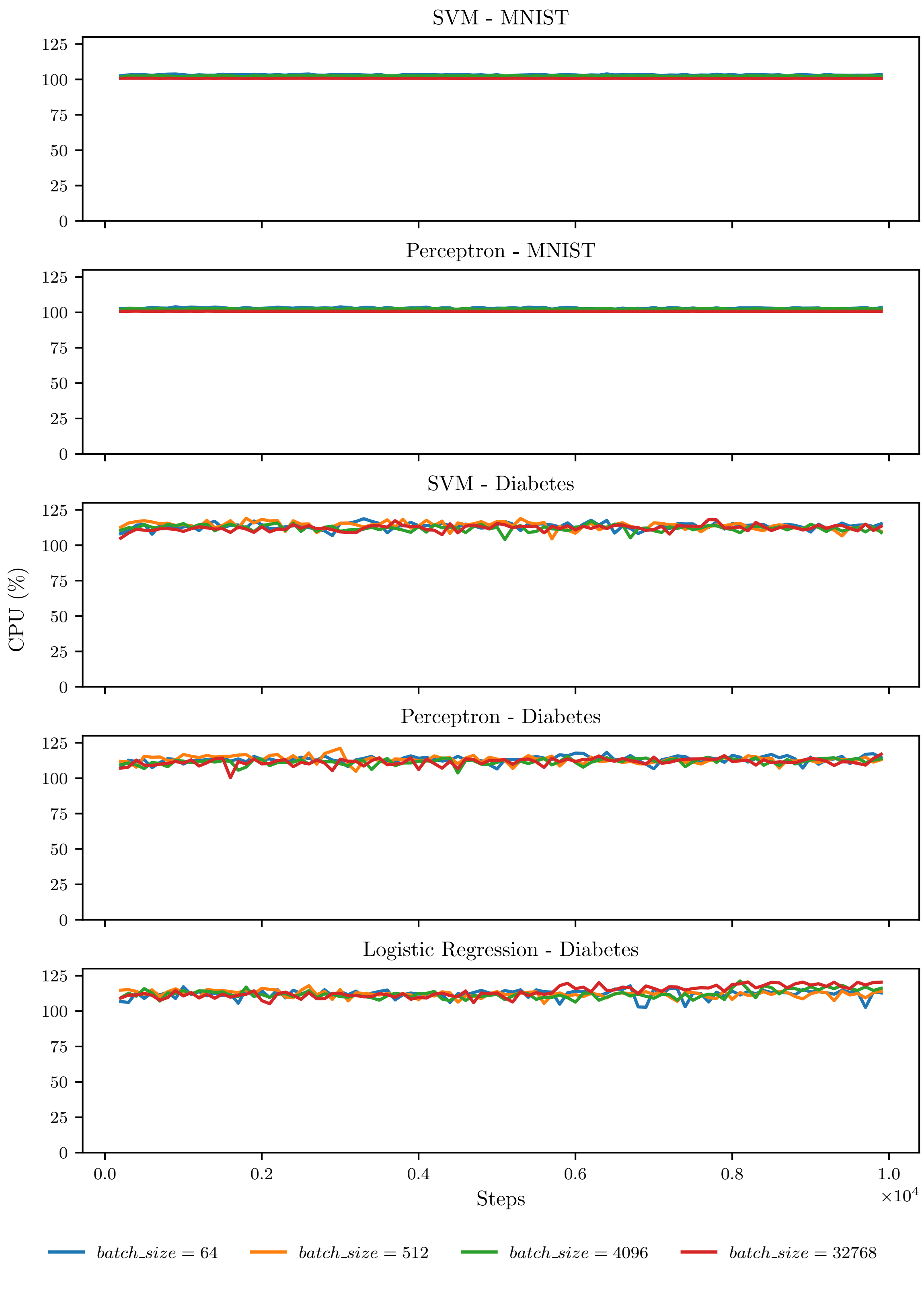

Figure 21. Effect of batch size on CPU utilization with classification workloads

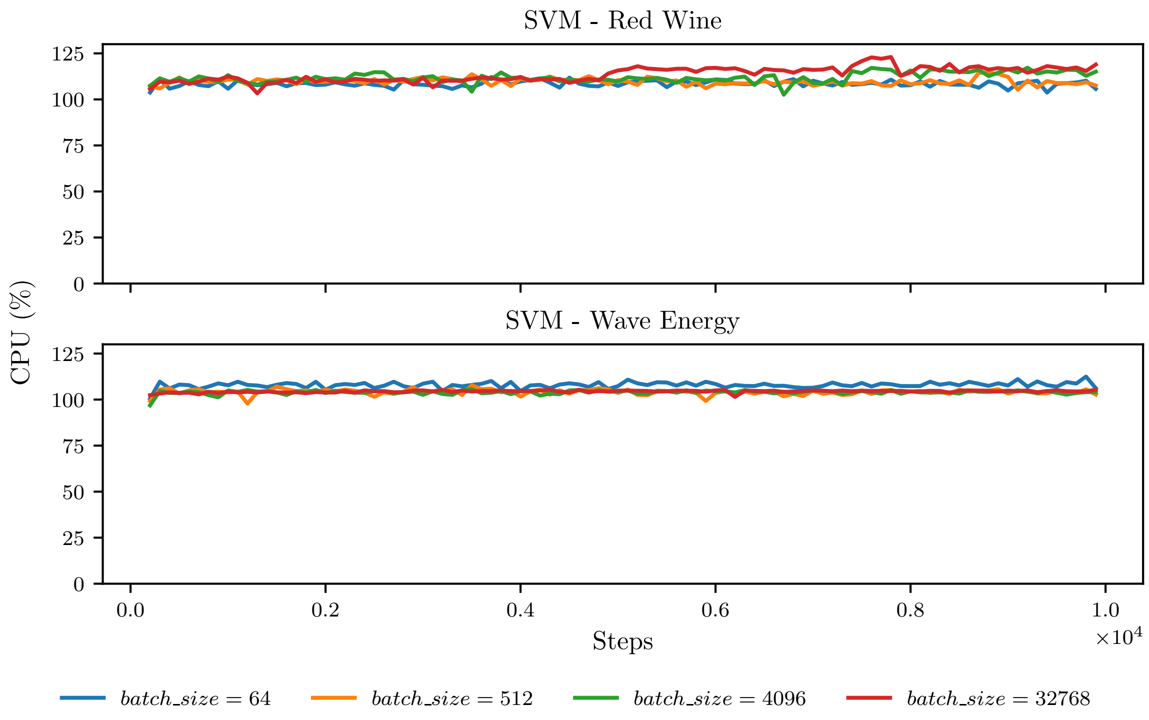

Figure 22. Effect of batch size on CPU utilization with regression workloads

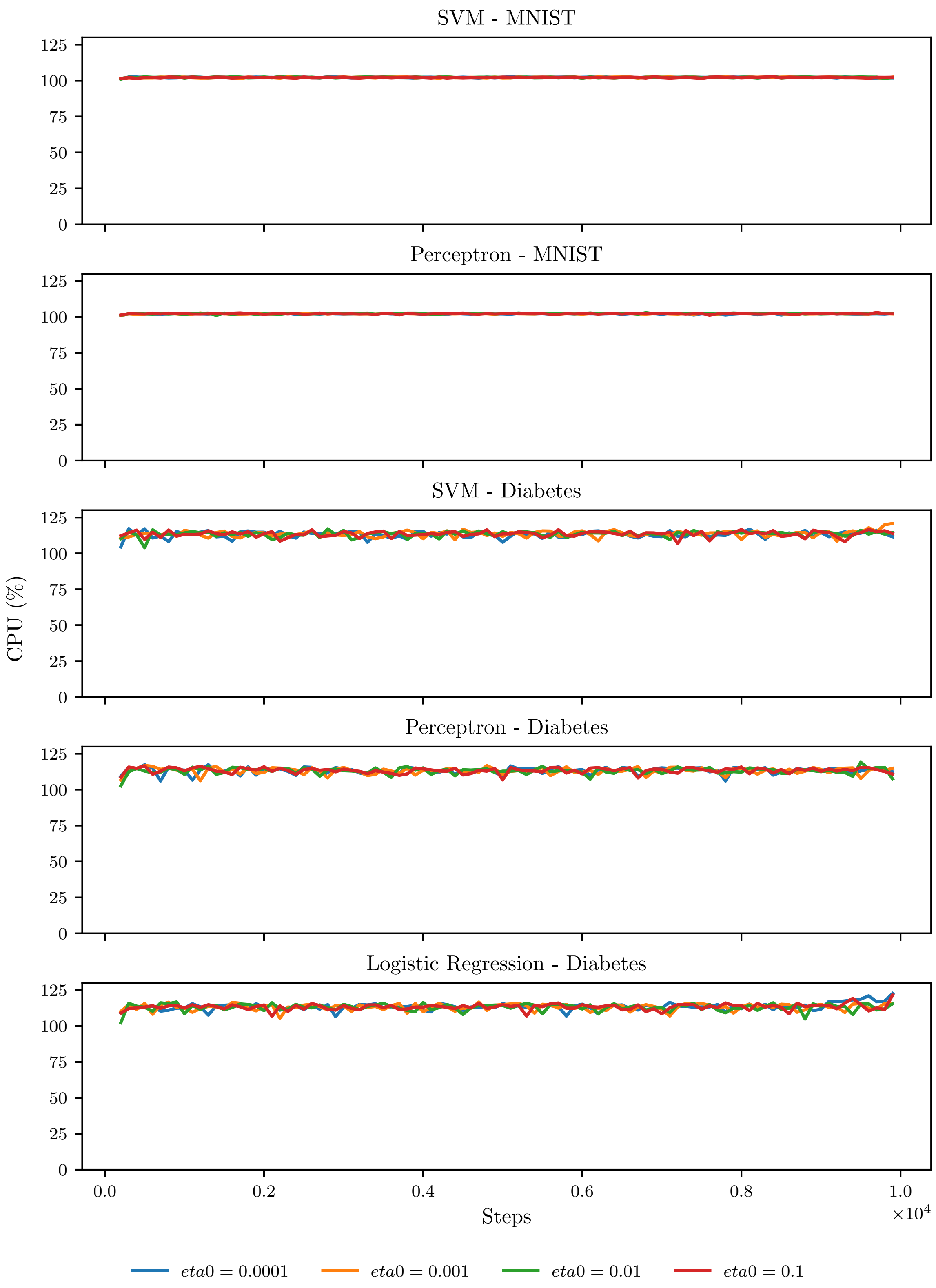

Figure 23. Effect of learning rate on CPU utilization with classification workloads

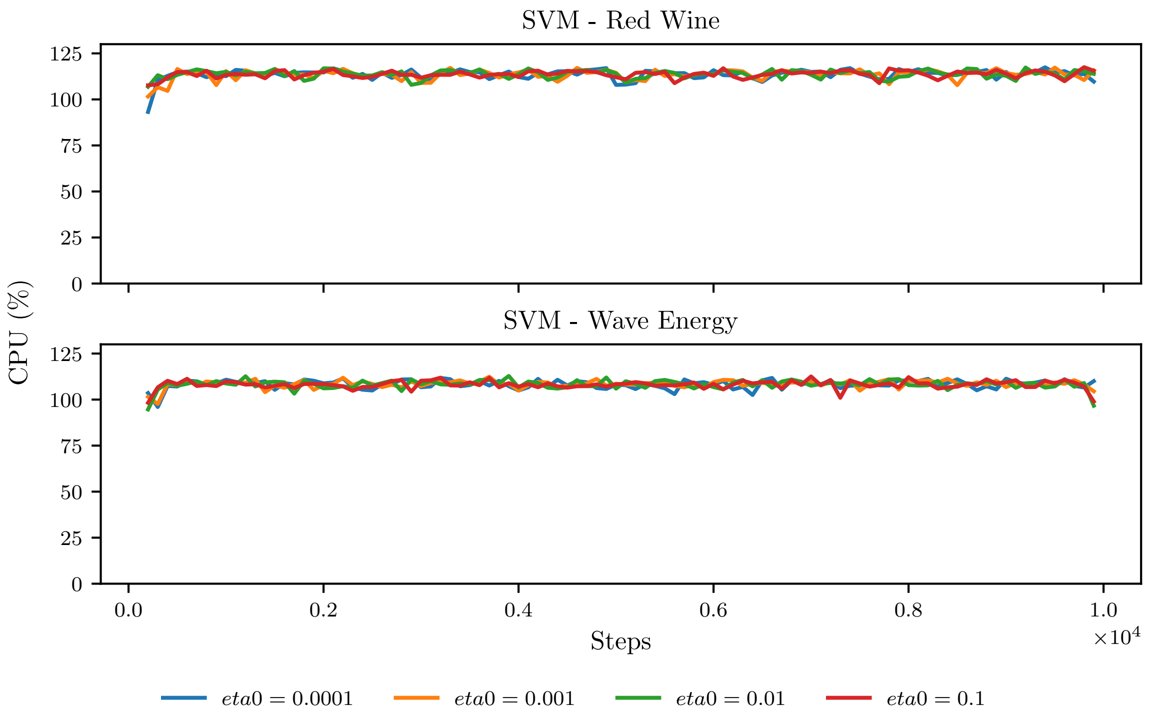

Figure 24. Effect of learning rate on CPU utilization with regression workloads

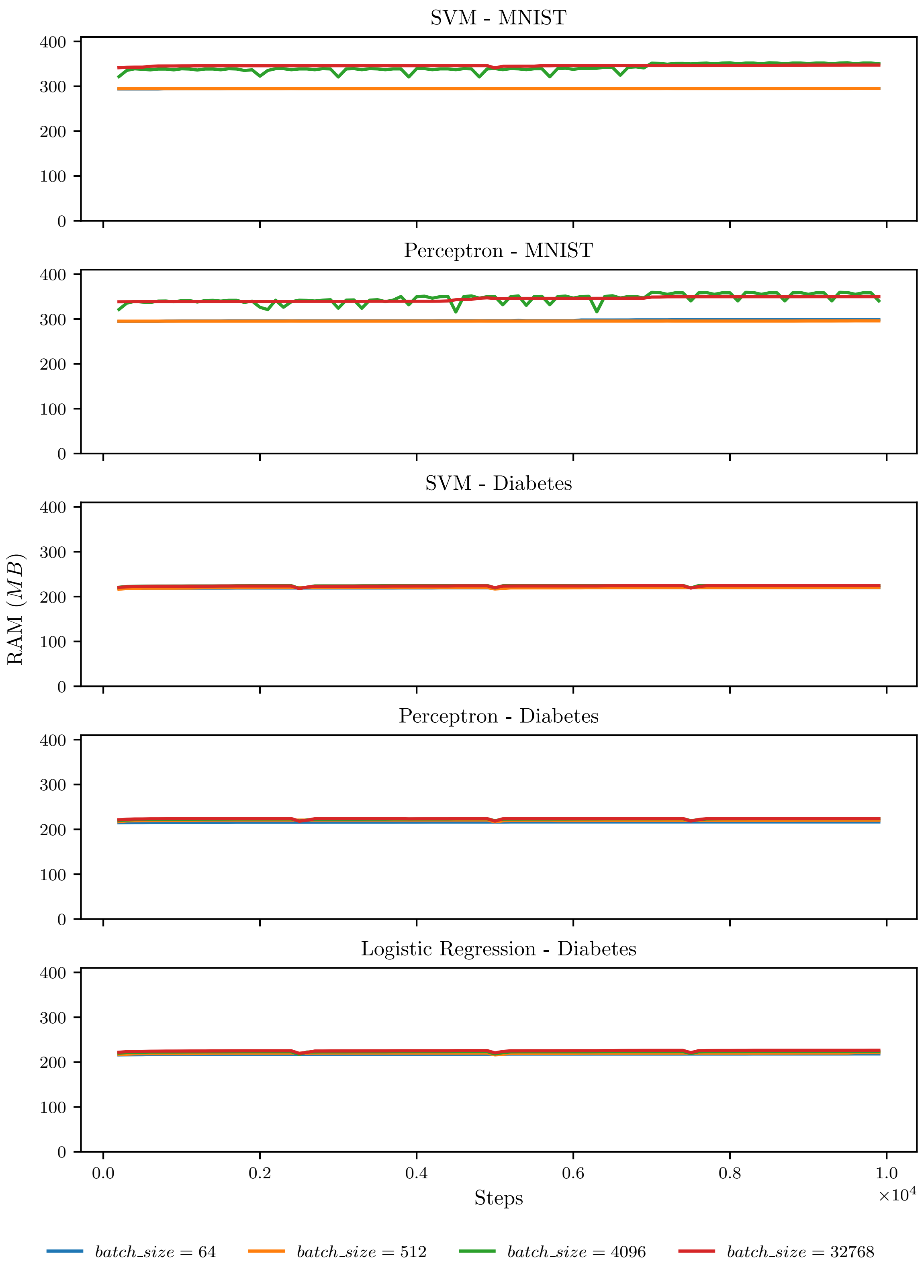

Figure 25. Effect of batch size on memory utilization with classification workloads

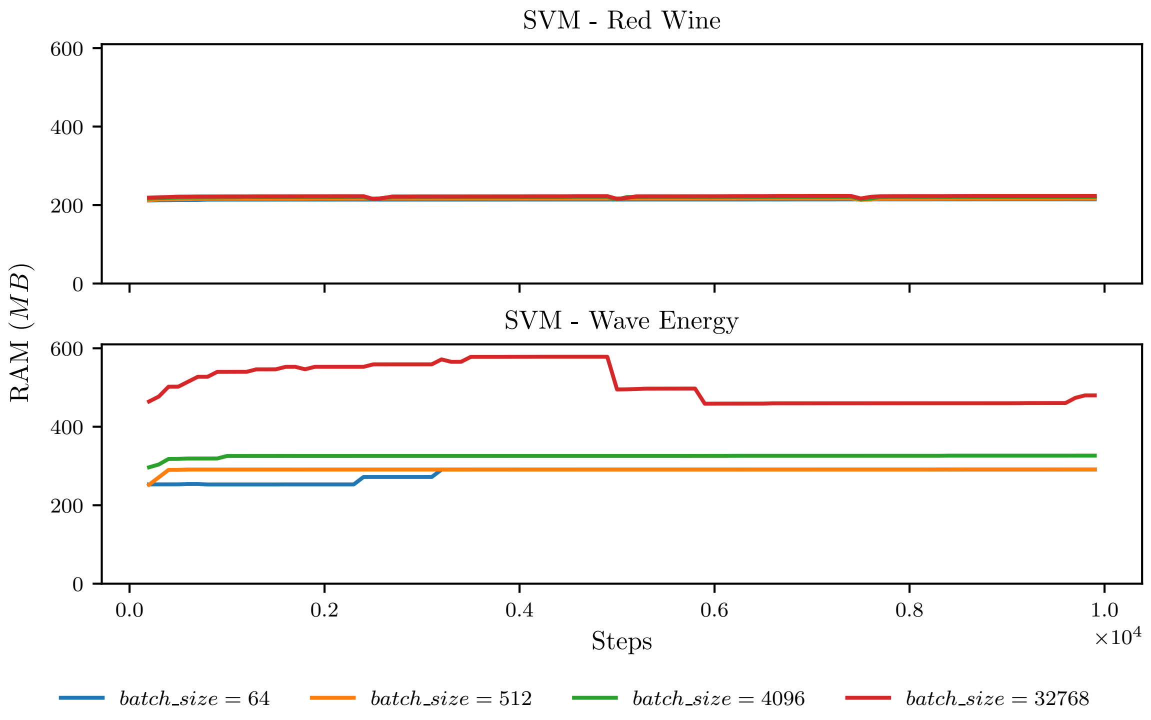

Figure 26. Effect of batch size on memory utilization with regression workloads

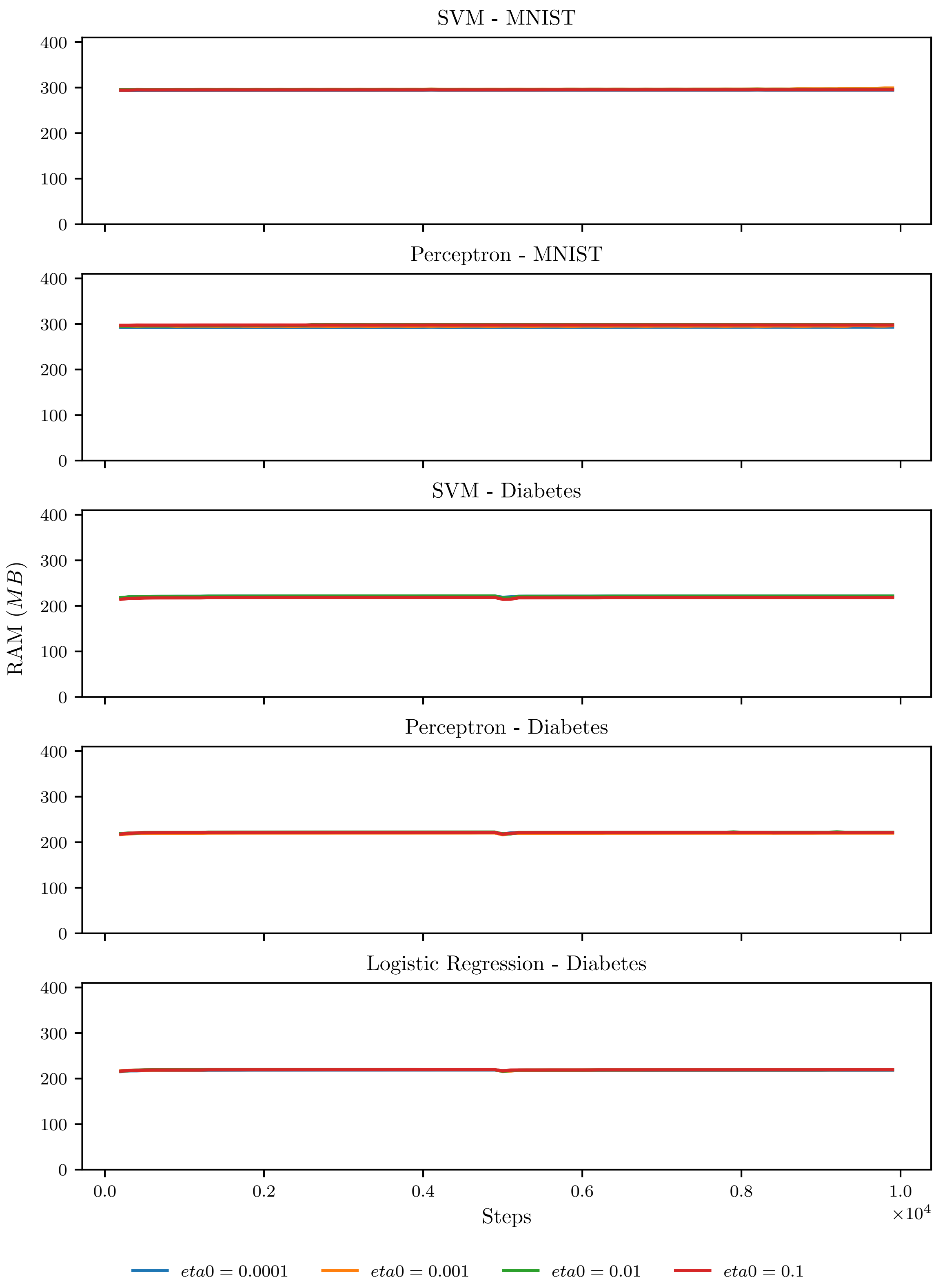

Figure 27. Effect of learning rate on memory utilization with classification workloads

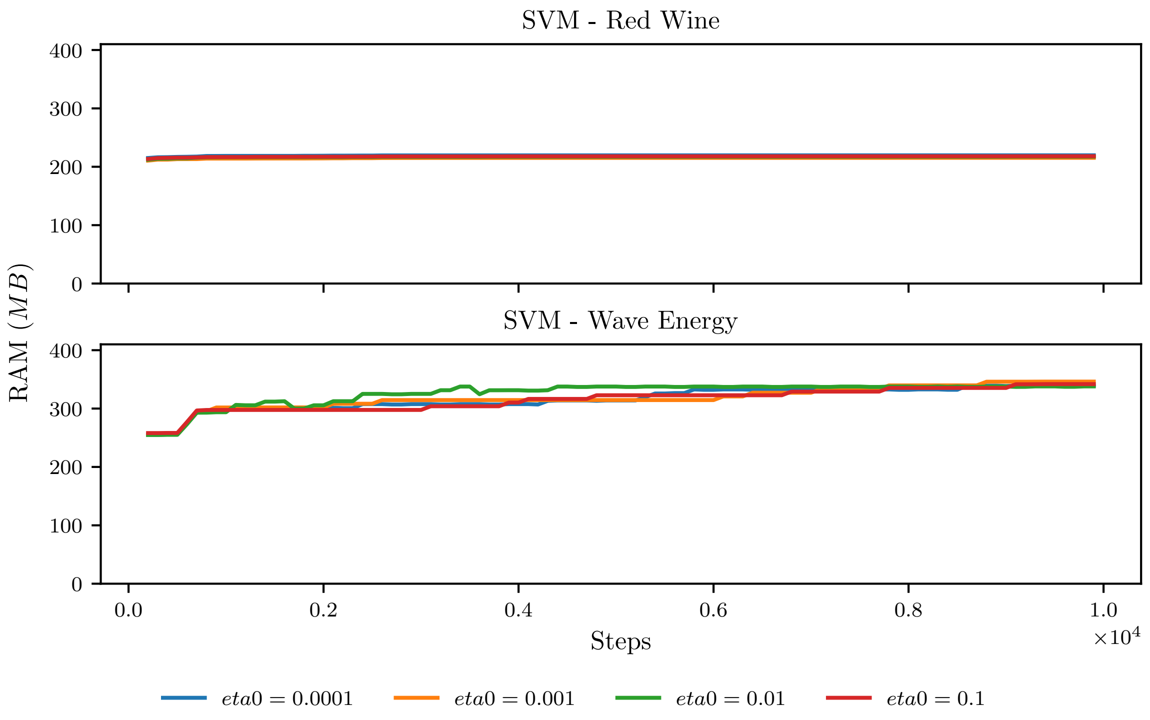

Figure 28. Effect of learning rate on memory utilization with regression workloads

4 Discussion

After presenting the results of the experiments, the next step is to interpret them and put them into context while considering limitations. Section 4.1 revisits and answers each research question in detail. Section 4.2 brings together the interpretation and introduces implications for research and practice. Section 4.3 addresses the limitations, and section 4.4 discusses future work.

4.1 Research Questions revisited

Research questions are answered by revisiting the research questions in the context of the results. Each research question is discussed separately in detail.

4.1.1 RQ1: How does system performance change over time during model training?

System performance remained constant within an individual training run, but there were differences between runs. In particular, there was a sensitivity to workloads with large differences if the dataset or, in some cases, the algorithm was different.

Mean step time was much higher for workloads using the MNIST dataset compared to the rest of the workloads. This was not a surprising result, as the MNIST data consists of images represented by numerous features, resulting in more computation necessary for each training step. The level of CPU utilization percentage seemed to be dependent on the time the run was performed and not on the workload. Surprisingly, it was not a reliable or a useful metric for measuring compute efficiency during model training.

Similarly, memory utilization was larger for workloads with larger datasets, but the difference was not as drastic as with mean step time. The workloads chosen were not particularly memory intensive and the Python process overhead contributed the most to the total process memory utilization.

4.1.2 RQ2: How do changes in hyperparameters affect system performance during model training?

Changes in learning rate did not affect system performance metrics either with classification or regression workloads. In contrast, workloads with large datasets were sensitive to large batch sizes, while workloads with small datasets were not.

The difference in mean step time with different batch sizes was not very noticeable when the chosen batch size was small and very noticeable when the chosen batch size was large, assuming the workload had a large dataset. As with RQ1, CPU utilization was not a useful metric even with different hyperparameters. Memory utilization grew with large batch size if the workload had a large dataset. In particular, the largest dataset, MNIST, had the largest effect on mean step time and memory utilization with an increased batch size.

4.1.3 RQ3: How does early stopping on system performance criteria affects the compute budget during model training?

The model's system performance could be predicted without fully training it as the system performance did not change during training. Large batch sizes increased mean step time and memory utilization on workloads with large datasets.

The amount of training required could potentially be reduced if early stopping was used based on system performance criteria. For example, if the system performance criteria are to use less than 400MB of memory, then runs that exceed this threshold will be early stopped and not consume the total compute budget.

4.2 Interpretation

The results are interpreted in two parts. In the first part the discussion focuses on implications for research in an academic context. The second part of the discussion focuses on implications for practice in an industrial or commercial context.

4.2.1 Implications for research

Cardoso Silva et al. (2020) in their paper identify key system metrics for monitoring a production machine learning system. Key metrics identified include task completion time, CPU and GPU usage, memory usage, disk input/output, and network traffic, and their collection was implemented in a tool called *Ubenchmark* (Cardoso Silva et al. 2020). The researchers, in particular, focus on empirically monitoring and performance benchmarking of the machine learning system.

This thesis builds upon existing literature with contributions related to measuring system performance metrics during machine learning model training. The first contribution is measuring system performance metrics, including mean step time during model training on different workloads. Mean step time is an important metric that is more informative than CPU utilization percentage when evaluating compute efficiency. Having a baseline on how different system performance metrics behave during model training allows for comparing results and has implications for planning experiments, hyperparameter optimization and techniques like early stopping.

The second contribution is measuring the effects of hyperparameters on system performance. Not all hyperparameters affect every metric, and there is a certain amount of overhead, which is constant and does not depend on the value of the hyperparameter. Measuring which hyperparameters affect system performance allows us to focus on hyperparameters with the biggest impact if compute budget is an issue. This can be further applied to setting criteria for early stopping, such as maximum memory or maximum mean step time.

The final contribution is introducing the concept of using system performance metrics for early stopping during hyperparameter optimization. Large datasets might stretch hardware capabilities and setting resource constraints can be necessary for efficient experimentation. For example constraining memory utilization might allow for training several models in parallel on different processor cores or on hardware with a limited amount of memory.

4.2.2 Implications for practice

Measuring system performance metrics during model training allows for debugging and performance optimization during development. In addition, during real-world operation of machine learning systems monitoring system performance metrics allows for planning for adequate model training hardware and detecting anomalies during model training. In particular, knowing that system performance metrics remain mostly constant during model training allows for performance prediction without wasting resources to fully train the model, resulting in reduced costs. It is also possible to check early during the training process whether the model will fit resource constraints. Counterintuitively, CPU utilization was not a useful metric, and metrics such as mean step time should be used for measuring compute efficiency of model training.

Real-world training of machine learning systems will benefit from using fewer system resources for similar results. Especially with cloud computing it is possible to save on compute costs by reducing memory utilization and picking smaller and cheaper server instances. It is also important from a development perspective to determine thresholds after which there are diminishing results. Increasing the batch size for example might result in larger memory utilization and longer mean step time, but not affect accuracy or test loss.

Early stopping without training the model to completion can be a cost-saving measure and allow for either a smaller compute budget or the use of the compute budget more efficiently. For example, by early stopping when the mean step time grows without the test loss converging faster, the saved model training steps can be used with a different hyperparameter configuration in which the test loss converges faster.

4.3 Limitations

While the reliability is good due to the computational and repeatable nature of experiments, available resources and the scope of the thesis resulted in some trade-offs that affect internal and external validity.

The main issues with the experiments are a lack of variety and a small sample of workloads and metrics. Increasing the sample size of datasets evaluated would make the required amount of compute resources considerably larger. The lack of variety is a more difficult problem to tackle as choosing more realistic datasets would require more sophisticated preprocessing steps, making the results depend on arbitrary choices and becoming less reproducible and less useful.

The requirement to collect metrics between batches, which requires support for continuing training, is not always supported in readily available implementations. This, in practice, limited the selection to a handful of algorithms based on stochastic gradient descent and might not be applicable to other types of algorithms. Metrics also suffered from a similar limitation as not all process-level metrics are easily available from Python and some metrics, such as disk or network usage, are not applicable to the performed experiments.

Limited compute resources also affected the ability to run different hyperparameter configurations. Testing many more hyperparameter configurations would allow the establishment of a trend line and overhead thresholds, which would be a useful tool for performance prediction. At the same time this would require more compute resources.

4.4 Future Work

In addition to early stopping during hyperparameter tuning, other techniques related to AutoML, such as performance prediction and neural architecture search, could also be applied to system performance metrics. This would potentially allow for more efficient compute budget allocation and for faster experimentation.

The machine learning practice community would benefit from more thorough research into system performance metrics and reporting of metrics in machine learning research. Currently, if published benchmarks and comparisons do not include system performance metrics, additional experimentation or search of alternative sources of information is required.

5 Conclusions

The aim of this thesis was to investigate whether using early stopping with system metrics leads to more efficient hyperparameter tuning when there are resource constraints. Experiments focused on training machine learning models based on stochastic gradient descent on a small set of benchmark datasets and recording system performance metrics during model training. This thesis offers contributions to both research and practice.

Investigating system performance metrics during model training provided several insights. System performance remained constant during model training allowing for better planning of experiments during research by predicting system performance without fully training the model. In practice this would allow for tailoring model training to for example reduce hardware costs or fit within resource constraints.

Bibliography

Abadi, Martín, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2016. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,* arXiv:1603.04467. Visited on March 5, 2024. https://doi.org/10.48550/arXiv.1603.04467. arXiv: 1603.04467 [cs].

Amodei, Dario, and Danny Hernandez. 2018. *AI and Compute.* https://openai.com/research/ai-and-compute. Visited on July 29, 2023.

Aslanpour, Mohammad S., Sukhpal Singh Gill, and Adel N. Toosi. 2020. "Performance Evaluation Metrics for Cloud, Fog and Edge Computing: A Review, Taxonomy, Benchmarks and Standards for Future Research". *Internet of Things* 12:100273. ISSN: 2542-6605, visited on March 26, 2024. https://doi.org/10.1016/j.iot.2020.100273.

Baker, Bowen, Otkrist Gupta, Ramesh Raskar, and Nikhil Naik. 2017. *Accelerating Neural Architecture Search Using Performance Prediction,* arXiv:1705.10823. Visited on January 25, 2023. https://doi.org/10.48550/arXiv.1705.10823. arXiv: 1705.10823 [cs].

Bischl, Bernd, Giuseppe Casalicchio, Matthias Feurer, Pieter Gijsbers, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. 2017. *OpenML Benchmarking Suites.* https://arxiv.org/abs/1708.03731v3. Visited on November 16, 2023.

Breck, Eric, Shanqing Cai, Eric Nielsen, Michael Salib, and D. Sculley. 2017. "The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction". In *Proceedings of IEEE Big Data.*

Brunnert, Andreas, Andre van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, et al. 2015. *Performance-Oriented DevOps: A Research Agenda,* arXiv:1508.04752. Visited on January 17, 2023. https://doi.org/10.48550/arXiv.1508.04752. arXiv: 1508.04752 [cs].

Cabrera, Christian, Andrei Paleyes, Pierre Thodoroff, and Neil D. Lawrence. 2023. *Real-World Machine Learning Systems: A Survey from a Data-Oriented Architecture Perspective,* arXiv:2302.04810. Visited on March 4, 2024. https://doi.org/10.48550/arXiv.2302.04810. arXiv: 2302.04810 [cs].

Cardoso Silva, Lucas, Fernando Rezende Zagatti, Bruno Silva Sette, Lucas Nildaimon dos Santos Silva, Daniel Lucrédio, Diego Furtado Silva, and Helena de Medeiros Caseli. 2020. "Benchmarking Machine Learning Solutions in Production". In *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA),* 626–633. https://doi.org/10.1109/ICMLA51294.2020.00104.

Cawley, Gavin C, and Nicola L C Talbot. 2010. "On Over-fittingg inModel Selectionn and-Subsequent Selection Biass inPerformance Evaluation".

Chen, Andrew, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, et al. 2020. "Developments in MLflow: A System to Accelerate the Machine Learning Lifecycle". In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning,* 1–4. DEEM'20. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-8023-2, visited on October 24, 2023. https://doi.org/10.1145/3399579.3399867.

Chen, Jiasi, and Xukan Ran. 2019. "Deep Learning With Edge Computing: A Review". *Proceedings of the IEEE* 107 (8): 1655–1674. ISSN: 0018-9219, 1558-2256, visited on July 29, 2023. https://doi.org/10.1109/JPROC.2019.2921977.

Dai, Shuang, and Fanlin Meng. 2023. "Addressing Modern and Practical Challenges in Machine Learning: A Survey of Online Federated and Transfer Learning". *Applied Intelligence* 53 (9): 11045–11072. ISSN: 1573-7497, visited on March 7, 2024. https://doi.org/10.1007/s10489-022-04065-3.

Domingos, Pedro. 2012. "A Few Useful Things to Know about Machine Learning". *Communications of the ACM* 55 (10): 78–87. ISSN: 0001-0782, visited on March 14, 2023. https://doi.org/10.1145/2347736.2347755.

Fernandez-Lozano, Carlos, Marcos Gestal, Cristian R. Munteanu, Julian Dorado, and Alejandro Pazos. 2016. “A Methodology for the Design of Experiments in Computational Intelligence with Multiple Regression Models”. *PeerJ* 4:e2721. ISSN: 2167-8359, visited on February 15, 2023. https://doi.org/10.7717/peerj.2721.

Finzer, William. 2013. “The Data Science Education Dilemma”. *Technology Innovations in Statistics Education* 7 (2). Visited on January 17, 2023. https://doi.org/10.5070/T572013891.

Fischer, Sebastian Felix, Matthias Feurer, and Bernd Bischl. 2023. “OpenML-CTR23 – A Curated Tabular Regression Benchmarking Suite”. In *AutoML Conference 2023 (Workshop).* Visited on November 16, 2023.

Forman, George, and Martin Scholz. 2009. “Apples-to-Apples in Cross-Validation Studies: Pitfalls in Classifier Performance Measurement”.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. *Distilling the Knowledge in a Neural Network,* arXiv:1503.02531. Visited on February 3, 2023. https://doi.org/10.48550/arXiv.1503.02531. arXiv: 1503.02531 `[cs, stat]`.

Hoffer, Elad, Itay Hubara, and Daniel Soudry. 2018. *Train Longer, Generalize Better: Closing the Generalization Gap in Large Batch Training of Neural Networks,* arXiv:1705.08741. Visited on August 2, 2023. https://doi.org/10.48550/arXiv.1705.08741. arXiv: 1705.08741 `[cs, stat]`.

Jabbari, Ramtin, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. “What Is DevOps? A Systematic Mapping Study on Definitions and Practices”. In *Proceedings of the Scientific Workshop Proceedings of XP2016,* 1–11. XP ’16 Workshops. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-4134-9, visited on March 7, 2024. https://doi.org/10.1145/2962695.2962707.

Janocha, Katarzyna, and Wojciech Marian Czarnecki. 2017. *On Loss Functions for Deep Neural Networks in Classification,* arXiv:1702.05659. Visited on August 27, 2023. https://doi.org/10.48550/arXiv.1702.05659. arXiv: 1702.05659 `[cs]`.

Kounev, Samuel, Klaus-Dieter Lange, and Jóakim Von Kistowski. 2020. *Systems Benchmarking: For Scientists and Engineers.* Cham: Springer International Publishing. ISBN: 978-3-030-41704-8 978-3-030-41705-5, visited on August 23, 2023. https://doi.org/10.1007/978-3-030-41705-5.

Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl. 2023. "Machine Learning Operations (MLOps): Overview, Definition, and Architecture". *IEEE Access* 11:31866–31879. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2023.3262138.

Le, Quoc V., Marc'Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg S. Corrado, Jeff Dean, and Andrew Y. Ng. 2012. *Building High-Level Features Using Large Scale Unsupervised Learning,* arXiv:1112.6209. Visited on February 3, 2023. https://doi.org/10.48550/arXiv.1112.6209. arXiv: 1112.6209 [cs].

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. 2015. "Deep Learning". *Nature* 521 (7553): 436–444. ISSN: 1476-4687, visited on June 15, 2023. https://doi.org/10.1038/nature14539.

Li, Yang, Kangbo Liu, Ranjan Satapathy, Suhang Wang, and Erik Cambria. 2023. *Recent Developments in Recommender Systems: A Survey,* arXiv:2306.12680. Visited on March 5, 2024. https://doi.org/10.48550/arXiv.2306.12680. arXiv: 2306.12680 [cs].

Liaw, Richard, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, and Ion Stoica. 2018. *Tune: A Research Platform for Distributed Model Selection and Training,* arXiv:1807.05118. Visited on February 22, 2023. https://doi.org/10.48550/arXiv.1807.05118. arXiv: 1807.05118 [cs, stat].

Lin, Tsung-Yi, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. 2014. "Microsoft COCO: Common Objects in Context". In *Computer Vision – ECCV 2014,* edited by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, 740–755. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-319-10602-1. https://doi.org/10.1007/978-3-319-10602-1_48.

Maclaurin, Dougal, David Duvenaud, and Ryan P. Adams. 2015. *Gradient-Based Hyperparameter Optimization through Reversible Learning,* arXiv:1502.03492. Visited on February 3, 2023. https://doi.org/10.48550/arXiv.1502.03492. arXiv: 1502.03492 `[cs, stat]`.

Mishra, Alok, and Ziadoon Otaiwi. 2020. "DevOps and Software Quality: A Systematic Mapping". *Computer Science Review* 38:100308. ISSN: 1574-0137, visited on January 17, 2023. https://doi.org/10.1016/j.cosrev.2020.100308.

Moreschi, Sergio, Gilberto Recupito, Valentina Lenarduzzi, Fabio Palomba, David Hastbacka, and Davide Taibi. 2023. *Toward End-to-End MLOps Tools Map: A Preliminary Study Based on a Multivocal Literature Review,* arXiv:2304.03254. Visited on March 24, 2024. https://doi.org/10.48550/arXiv.2304.03254. arXiv: 2304.03254 `[cs]`.

OpenAI. 2022. *Introducing ChatGPT.* https://openai.com/blog/chatgpt. Visited on July 24, 2023.

Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, et al. 2019. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In *Advances in Neural Information Processing Systems,* volume 32. Curran Associates, Inc. Visited on March 5, 2024.

Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python". *Journal of Machine Learning Research* 12 (85): 2825–2830. ISSN: 1533-7928, visited on October 25, 2023.

Perera, Pulasthi, Roshali Silva, and Indika Perera. 2017. "Improve Software Quality through Practicing DevOps". In *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer),* 1–6. https://doi.org/10.1109/ICTER.2017.8257807.

Posoldova, Alexandra. 2020. "Machine Learning Pipelines: From Research to Production". *IEEE Potentials* 39 (6): 38–42. ISSN: 1558-1772. https://doi.org/10.1109/MPOT.2020.3016280.

Prabhu, Ameya, Hasan Abed Al Kader Hammoud, Puneet Dokania, Philip H. S. Torr, Ser-Nam Lim, Bernard Ghanem, and Adel Bibi. 2023. “Computationally Budgeted Continual Learning: What Does Matter?” In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR),* 3698–3707. IEEE Computer Society. ISBN: 9798350301298, visited on March 7, 2024. https://doi.org/10.1109/CVPR52729.2023.00360.

Prechelt, Lutz. 1998. “Automatic Early Stopping Using Cross Validation: Quantifying the Criteria”. *Neural Networks* 11 (4): 761–767. ISSN: 0893-6080, visited on August 2, 2023. https://doi.org/10.1016/S0893-6080(98)00010-0.

Rasley, Jeff, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. “DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters”. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining,* 3505–3506. KDD ’20. New York, NY, USA: Association for Computing Machinery. ISBN: 978-1-4503-7998-4, visited on March 4, 2024. https://doi.org/10.1145/3394486.3406703.

Rodola, Giampaolo. 2023. *Giampaolo/Psutil.* Visited on November 2, 2023.

Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. *High-Resolution Image Synthesis with Latent Diffusion Models,* arXiv:2112.10752. Visited on July 22, 2023. https://doi.org/10.48550/arXiv.2112.10752. arXiv: 2112.10752 `[cs]`.

Sarker, Iqbal H. 2021. “Machine Learning: Algorithms, Real-World Applications and Research Directions”. *SN Computer Science* 2 (3): 160. ISSN: 2661-8907, visited on March 7, 2024. https://doi.org/10.1007/s42979-021-00592-x.

Sculley, D., Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-François Crespo, and Dan Dennison. 2015. “Hidden Technical Debt in Machine Learning Systems”. In *Advances in Neural Information Processing Systems,* volume 28. Curran Associates, Inc. Visited on August 13, 2023.

Sergeev, Alexander, and Mike Del Balso. 2018. *Horovod: Fast and Easy Distributed Deep Learning in TensorFlow,* arXiv:1802.05799. Visited on March 5, 2024. https://doi.org/10.48550/arXiv.1802.05799. arXiv: 1802.05799 `[cs, stat]`.

Shallue, Christopher J., Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. 2019. “Measuring the Effects of Data Parallelism on Neural Network Training”. *Journal of Machine Learning Research* 20 (112): 1–49. ISSN: 1533-7928, visited on September 28, 2023.

Shankar, Shreya, Rolando Garcia, Joseph M. Hellerstein, and Aditya G. Parameswaran. 2022. *Operationalizing Machine Learning: An Interview Study,* arXiv:2209.09125. Visited on December 7, 2022. https://doi.org/10.48550/arXiv.2209.09125. arXiv: 2209.09125 [cs].

Shmueli, Galit. 2010. “To Explain or to Predict?” *Statistical Science* 25 (3). ISSN: 0883-4237, visited on March 14, 2023. https://doi.org/10.1214/10-STS330. arXiv: 1101.0891 [stat].

Smeds, Jens, Kristian Nybom, and Ivan Porres. 2015. “DevOps: A Definition and Perceived Adoption Impediments”. In *Agile Processes in Software Engineering and Extreme Programming,* edited by Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, 166–177. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. ISBN: 978-3-319-18612-2. https://doi.org/10.1007/978-3-319-18612-2_14.

Sokolova, Marina, and Guy Lapalme. 2009. “A Systematic Analysis of Performance Measures for Classification Tasks”. *Information Processing & Management* 45 (4): 427–437. ISSN: 0306-4573, visited on August 27, 2023. https://doi.org/10.1016/j.ipm.2009.03.002.

Stability AI. 2022. *Stable Diffusion Public Release.* https://stability.ai/blog/stable-diffusion-public-release. Visited on July 24, 2023.

Strubell, Emma, Ananya Ganesh, and Andrew McCallum. 2020. “Energy and Policy Considerations for Modern Deep Learning Research”. *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (09): 13693–13696. ISSN: 2374-3468, visited on July 29, 2023. https://doi.org/10.1609/aaai.v34i09.7123.

The Ray Team. *Memory Management — Ray 2.7.1.* https://docs.ray.io/en/latest/ray-core/scheduling/mem management.html. Visited on November 1, 2023.

Torralba, Antonio, and Alexei A. Efros. 2011. “Unbiased Look at Dataset Bias”. In *CVPR 2011,* 1521–1528. Colorado Springs, CO, USA: IEEE. ISBN: 978-1-4577-0394-2, visited on August 27, 2023. https://doi.org/10.1109/CVPR.2011.5995347.

Touvron, Hugo, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, et al. 2023. *LLaMA: Open and Efficient Foundation Language Models,* arXiv:2302.13971. Visited on July 24, 2023. https://doi.org/10.48550/arXiv.2302.13971. arXiv: 2302.13971 [cs].

Vanschoren, Joaquin, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2014. “OpenML: Networked Science in Machine Learning”. *ACM SIGKDD Explorations Newsletter* 15 (2): 49–60. ISSN: 1931-0145, visited on November 16, 2023. https://doi.org/10.1145/2641190.2641198.

Waller, Jan, Nils C. Ehmke, and Wilhelm Hasselbring. 2015. “Including Performance Benchmarks into Continuous Integration to Enable DevOps”. *ACM SIGSOFT Software Engineering Notes* 40 (2): 1–4. ISSN: 0163-5948, visited on January 17, 2023. https://doi.org/10.1145/2735399.2735416.

Yang, Li, and Abdallah Shami. 2020. “On Hyperparameter Optimization of Machine Learning Algorithms: Theory and Practice”. *Neurocomputing* 415:295–316. ISSN: 09252312, visited on January 25, 2023. https://doi.org/10.1016/j.neucom.2020.07.061. arXiv: 2007.15745 [cs, stat].

Yu, Tong, and Hong Zhu. 2020. *Hyper-Parameter Optimization: A Review of Algorithms and Applications,* arXiv:2003.05689. Visited on January 10, 2024. https://doi.org/10.48550/arXiv.2003.05689. arXiv: 2003.05689 [cs, stat].

Zaharia, M., A. Chen, A. Davidson, A. Ghodsi, S. Hong, A. Konwinski, Siddharth Murching, et al. 2018. “Accelerating the Machine Learning Lifecycle with MLflow”. *IEEE Data Eng. Bull.,* visited on March 14, 2023.