CSCI 5525 - ADVANCED MACHINE LEARNING - FINAL PROJECT REPORT

Amirhossein Kiani, Roozbeh Ehsani, Rishikesh Joshi

1 INTRODUCTION

Optimization techniques form the cornerstone of machine learning, dictating the efficiency and success of training complex models. Among these techniques, Adam (2) has emerged as a dominant method, renowned for its adaptive learning rates and stability across a variety of tasks. Despite its widespread adoption, Adam is not without limitations, particularly its sensitivity to hyperparameter settings and occasional convergence issues in noisy gradient environments (3). In response to these challenges, advancements such as ADOPT (6) have been developed, offering enhanced convergence reliability by mitigating gradient correlation issues and improving momentum updates.

In parallel with the evolution of optimization strategies, the field of text modeling has witnessed groundbreaking developments, particularly with the advent of Long Short-Term Memory (LSTM) networks (4) and transformer-based models like BERT (1). LSTMs addressed the shortcomings of earlier recurrent neural networks (RNNs) by introducing mechanisms to retain long-term dependencies, which are critical for sequential data analysis. Transformers revolutionized text modeling further by leveraging self-attention mechanisms, enabling models to capture global context and dependencies more effectively than their predecessors.

The integration of advanced optimization techniques like ADOPT with state-of-the-art text modeling frameworks presents a unique opportunity to enhance model performance and training efficiency. This work explores the synergies between optimization strategies and text data modeling, focusing on practical implementations in LSTM and transformer architectures. By applying these methods to the Spooky Author Identification dataset, we aim to demonstrate improvements in convergence speed, model accuracy, and robustness.

The collaborative efforts of this project also emphasize the importance of adapting optimization strategies to specific tasks and data distributions. Through rigorous experimentation and iterative development, this research seeks to provide actionable insights into the optimization of modern machine learning systems, particularly in the domain of natural language processing (NLP).

2 PROBLEM OVERVIEW

Deep learning models have been using optimization methods based on gradients for training neural networks. While popular optimizers like ADAM, SGD, or more recent methods such as ADOPT are individually good at different aspects, no single optimizer is optimal across all training stages. For example, some optimizers are great at quickly driving down the loss early in training but struggle to reach lower final loss values, while others are more stable but slower to converge. The problem, therefore, is to determine how to intelligently switch between different optimizers during training to leverage their complementary strengths—improving both the convergence rate and the final accuracy or loss. In this project, we implement a reinforcement learning agent to dynamically switch between two optimizers during the training process of an LSTM in a text classification problem.

3 SIGNIFICANCE

This problem is interesting because choosing an appropriate optimizer is usually done manually and heuristically. This relies on prior experience rather than a principled, data-driven approach. By dynamically selecting optimizers, we can potentially achieve faster convergence early on and still enjoy the fine-grained refinements of a slower but more stable optimizer later. This approach can reduce the overall training time, improve the final model performance, and make optimization a

more automated process—ultimately leading to more efficient use of computational resources and potentially better models.

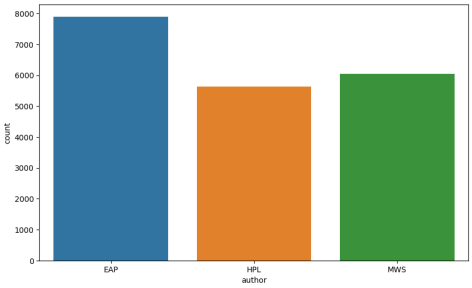
4 Dataset Overview

The **Spooky Author Identification Dataset** is a collection of texts designed to identify authorship based on writing style. It focuses on three prominent 19th and early 20th-century writers known for their eerie and atmospheric literature:

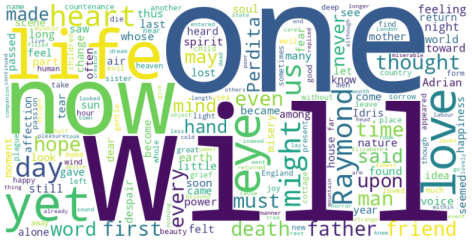
- **EAP – Edgar Allan Poe:** An American writer known for his poetry and short stories, often exploring themes of death, mystery, and the macabre.
- **HPL – H. P. Lovecraft:** Best known for his contributions to the horror fiction genre, particularly cosmic horror and the mythos of ancient, otherworldly beings.
- **MWS – Mary Shelley:** A versatile author known for works spanning novels, dramas, travel writing, and biographies, most notably the pioneering science fiction novel *Frankenstein*.

4.1 Exploratory Data Analysis (EDA) and Wordclouds

Exploratory Data Analysis (EDA) involves summarizing the key characteristics of the dataset through visualization and statistical analysis to better understand its structure. **Wordclouds** are visual representations of the most frequent words in the dataset, where word size reflects their frequency, providing insight into common themes and patterns. Here we have visualized in figure1 some basic statistics in the data, like the distribution of entries for each author and wordcloud for the author "MWS" which highlights prominent words that frequently appear in her texts. To ensure the wordcloud focuses on meaningful words, we have removed common words like *"he," "it," "the,"* and *"and"* that frequently appear across all contexts.



(a) Histogram of Author Distribution



(b) Wordcloud for MWS

Figure 1: Exploratory Data Analysis

5 Model Selection: RNN and LSTM

Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) models were used for text classification to identify the author based on writing style.

5.1 Recurrent Neural Network (RNN)

A Recurrent Neural Network (RNN) is a type of neural network designed for processing sequential data. Each tokenized word is embedded into a vector of arbitrary size, denoted as x_t, and used as input to the model at each time step. It maintains a memory of previous inputs through hidden states h_{t-1}, allowing it to capture temporal dependencies. RNNs are commonly used for tasks such as language modeling, text classification, and speech recognition. Full details of the models are provided in the code comments.

5.2 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) network is an advanced type of RNN designed to address the vanishing gradient problem. Each tokenized word is embedded into a vector of arbitrary size, denoted as x_t, and used as input at each time step. LSTMs use special memory cells with gates (input, forget, and output) to selectively remember or forget information, enabling them to capture long-term dependencies. They are widely used for tasks like text classification, language modeling, and sequence prediction. Full details of the models are provided in the code comments.

6 Enhancing Optimizers Using Reinforcement Learning

6.1 Multi-Armed Bandits

Multi-armed bandits form a class of reinforcement learning problems centered around decision-making under uncertainty. The name is the analogy of a gambler faced with multiple slot machines ("one-armed bandits"), each with an unknown payout rate. The gambler's objective is to maximize their cumulative reward by not only exploiting the machine that seems most promising based on past experiences but also exploring the others to avoid missing out on potentially higher payouts. A multi-armed bandit model formally provides a set of choices (arms), and each time an arm is chosen, it yields a reward drawn from an unknown distribution. Over time, the learner updates its estimates of each arm's value and follows an explore-exploit strategy that balances the need to acquire information with the need to maximize returns, eventually converging to a policy that optimizes long-term returns. In this problem, we prioritized high accuracy/low loss and convergence rate in the reward function.

6.2 Approach

Our approach leverages reinforcement learning, specifically the multi-armed bandit framework, to dynamically select between two optimizers (ADOPT and SGD) at each epoch of the LSTM training on text classification. Instead of committing to one optimizer, we model each optimizer as an "arm" of the bandit. Initially, we pre-trained our MAB on the HuggingFace IMDB dataset (5) for 30 epochs using an LSTM model. This first step enabled the bandit to learn baseline Q-values of the candidate optimizers, ADOPT and SGD, through observing which optimizer yielded higher improvement in validation over time.

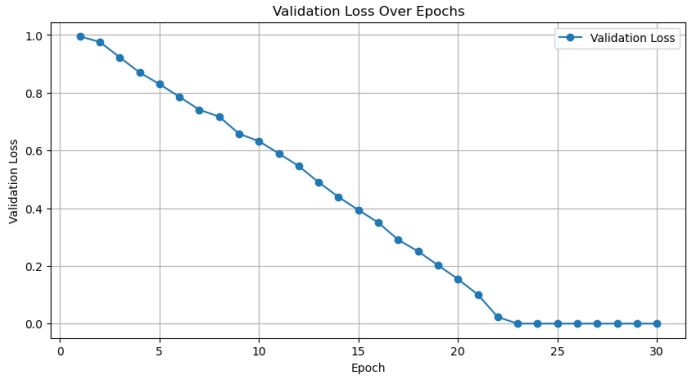
Once the binary MAB is pre-trained, it can then be applied to new tasks. At each epoch of any new model training process, the bandit selects an optimizer based on its current Q-values' probability distribution. We then measure the resulting improvement in validation loss or accuracy after applying the chosen optimizer for that epoch and treat it as a reward. This creates a feedback loop that updates the Q-values of MAB even on new tasks. Over time, as the bandit receives rewards across different problems, it learns which optimizer tends to be more effective under various training conditions.

6.3 Analysis

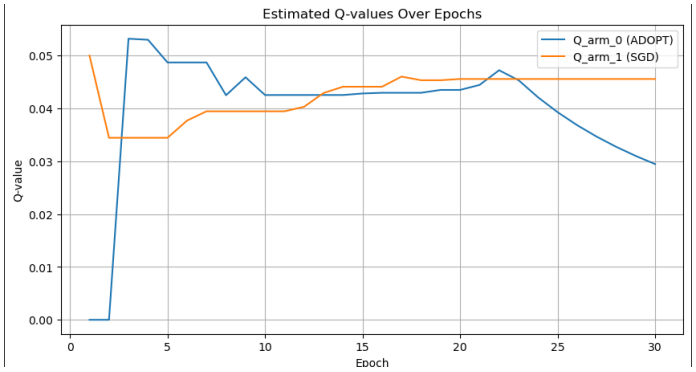
In the following plots, we show pre-training for our MAB approach on the IMDB dataset, and the results of using the pre-trained MAB on the Spooky Author Identification dataset. The sub-figure (a) demonstrates how the validation loss decreases through 30 epochs and how the bandit has learned to choose optimizers that continuously improve the performance of the model. Similarly, in sub-figure (b), we show the bandit's Q-values at each time step during pre-training. Averaging these over time yields converging Q-values that reflect the bandit's growing confidence in which optimizer (arm) is better. Taken together, these results suggest that a bandit pre-trained on IMDB data emerges well-equipped to tackle the Spooky Author Identification dataset with informed and adaptive optimization strategies.

In sub-figure (c), we see the validation loss for the Spooky Author Identification dataset over just 5 epochs using two approaches: ADOPT alone and the pre-trained multi-armed bandit (MAB) selecting between SGD and ADOPT. While pure ADOPT is known for its fast convergence and theoretical guarantees, the MAB combination has a slightly faster reduction in loss. Notably, in the second and third epochs, the bandit chose the SGD arm, causing a more pronounced early drop in loss. Af-

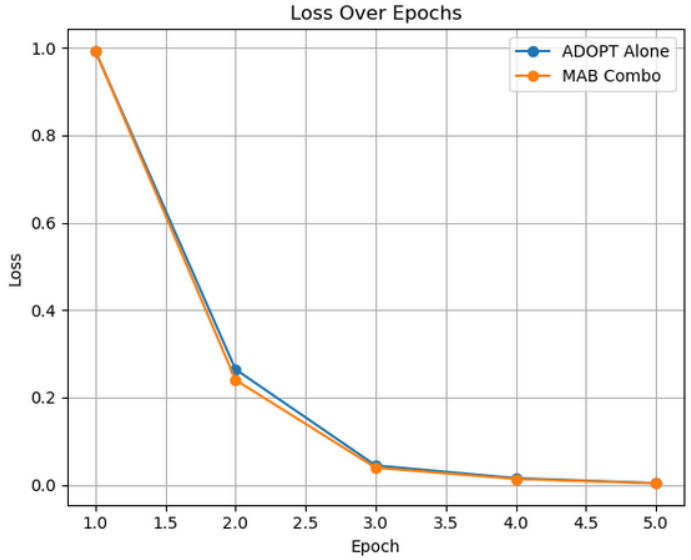
terwards, the selection shifted back to ADOPT, guiding the training to an even lower final loss. These results suggest that a pre-trained bandit strategy can effectively leverage the strengths of both optimizers to improve convergence beyond what ADOPT alone can achieve in a limited training window.



(a) MAB training on IMDB: Validation loss over 30 epochs



(b) Training Q-values on IMDB Dataset



(c) Pre-trained MAB (SGD+ADOPT) vs. ADOPT alone

7 CODE

This section provides the general overview of the code setup.

7.1 LSTM

```
class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, num_classes, max_len, pretrained_embeddings=None, freeze_embeddings=False):
        super(LSTMClassifier, self).__init__()

        # Embedding layer (with optional pretrained embeddings)
        if pretrained_embeddings is not None:
            self.embedding = nn.Embedding.from_pretrained(pretrained_embeddings, freeze=freeze_embeddings)
        else:
            self.embedding = nn.Embedding(vocab_size, embedding_dim)

        # LSTM layer with dropout and recurrent dropout
        self.lstm = nn.LSTM(input_size=embedding_dim,
                            hidden_size=hidden_dim,
                            batch_first=True,
                            dropout=0.3)  # Applies dropout between LSTM layers (when num_layers > 1)

        # Dense layer for classification
        self.fc = nn.Linear(hidden_dim, num_classes)

        # Softmax activation(We do not use it since nn.CrossEntropy does have sofmax layer in itself)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, x):
        # x shape: (batch_size, max_len)
        x = self.embedding(x)  # Shape: (batch_size, max_len, embedding_dim)

        # LSTM forward pass
        lstm_out, (h_n, c_n) = self.lstm(x)  # h_n shape: (1, batch_size, hidden_dim)

        # Use the last hidden state for classification
        final_hidden_state = h_n[-1]  # Shape: (batch_size, hidden_dim)

        # Dense layer and softmax activation
        out = self.fc(final_hidden_state)  # Shape: (batch_size, num_classes)

        return out

# Model hyperparameters
vocab_size = len(word_index) + 1  # Vocabulary size
embedding_dim = 300               # Embedding dimension
hidden_dim = 100                  # LSTM hidden units
num_classes = 3                   # Number of output classes
max_len = 1500                    # Maximum sequence length

# Initialize the model
model_LSTM_ADOPT = LSTMClassifier(vocab_size, embedding_dim, hidden_dim, num_classes, max_len)

# Print the model summary
print(model_LSTM_ADOPT)

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizeradop = ADOPT(model_LSTM_ADOPT.parameters(), lr=1e-3)
```

```
optimizeradam = torch.optim.Adam(model_LSTM_ADOPT.parameters(), lr=1e-3)
```

Listing 1: LSTM Setup

7.2 Multi-Armed Bandits

```
num_arms = 2
epsilon = 0.2
Q = np.zeros(num_arms)
N = np.zeros(num_arms)

rewards_history = []
chosen_arms_history = []
Q_history = []

def choose_arm():
    if np.random.rand() < epsilon:
        return np.random.randint(num_arms)
    else:
        return np.argmax(Q)

def update_Q(arm, reward):
    N[arm] += 1
    alpha = 1.0 / N[arm]
    Q[arm] = Q[arm] + alpha * (reward - Q[arm])

def train_one_epoch(model, loader, optimizer, criterion):
    model.train()
    total_loss = 0.0
    for inputs, labels in loader:
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss/len(loader)

def evaluate_loss(model, loader, criterion):
    model.eval()
    total_loss = 0.0
    with torch.no_grad():
        for inputs, labels in loader:
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            total_loss += loss.item()
    return total_loss / len(loader)

num_epochs = 10
model = LSTMClassifier(vocab_size, embedding_dim, hidden_dim, num_classes
    , max_len)
criterion = torch.nn.CrossEntropyLoss()

def adopt_optimizer_fn(params):
    return ADOPT(params, lr=1e-3)

def sgd_optimizer_fn(params):
    return torch.optim.SGD(params, lr=1e-3)

previous_val_loss = evaluate_loss(model, val_loader, criterion)

for epoch in range(num_epochs):
    arm = choose_arm()

```

```
    if arm == 0:
        optimizer = adopt_optimizer_fn(model.parameters())
        chosen_optimizer = "ADOPT"
    else:
        optimizer = sgd_optimizer_fn(model.parameters())
        chosen_optimizer = "SGD"

    # Train one epoch
    train_loss = train_one_epoch(model, train_loader, optimizer,
    criterion)

    # Evaluate after training step
    current_val_loss = evaluate_loss(model, val_loader, criterion)

    # Compute reward (improvement in val loss)
    reward = (previous_val_loss - current_val_loss)

    # Update bandit Q-values
    update_Q(arm, reward)

    previous_val_loss = current_val_loss
```

Listing 2: Multi-Armed Bandits Setup

8 Future Directions

There are several ways this work can be extended and refined in the future. First, the multi-armed bandit framework can be generalized to include a wider variety of optimizers than the two considered here, ADOPT and SGD, thus providing a richer set of ”arms” for the bandit to explore and exploit. This could include adaptive methods, such as RMSProp and Nadam, or newer optimizers tailored to specific model architectures. Further balancing the exploration and exploitation could be obtained by including more sophisticated bandit algorithms such as UCB or Thompson Sampling. Another promising direction could be an application of this approach to larger-scale models and tasks, including transformer-based architectures, such as BERT, or domain-specific datasets. This would finally allow the integration of hyperparameter tuning or automated curriculum learning techniques into a more holistic, fully automated training pipeline that would adjust optimizers, learning rates, and even training sequences on the fly, pushing the boundaries of model performance and efficiency. For improved results, it would also be better to model this as a Markov environment and use more complex algorithms like PPO or Q-Learning to learn the intricacies of such optimizers and when to pick them, instead of the naive multi-armed bandit approach taken in this paper.

9 Conclusion

In this work, we investigated the use of a multi-armed bandit framework for dynamic selection between different optimizers, namely ADOPT and SGD, while training an LSTM model on a text classification task. Our experiments showed that even a small amount of pre-training on the IMDB dataset can guide the bandit to make slightly more effective choices than relying on a single optimizer like ADOPT, leading to marginal but noticeable gains in convergence speed.

Although the observed gains are modest, this result is an interesting proof of concept. It suggests that with longer pre-training, richer feedback, or a wider range of choices over optimizers, the MAB's policy could become increasingly fine-grained, leading to greater performance gains. Moreover, there is nothing intrinsic in the approach that limits its application to LSTMs or to one text classification task. The same principle could apply to other architectures, ranging from standard RNNs to transformer-based models such as BERT, dynamically leveraging various optimization strategies to adapt to a range of training conditions. Moreover, this is definitely not restricted to merely deciding between two optimizers.

REFERENCES

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[4] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[5] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification?, 2020.

[6] Shohei Taniguchi, Keno Harada, Gouki Minegishi, Yuta Oshima, Seong Cheol Jeong, Go Nagahara, Tomoshi Iiyama, Masahiro Suzuki, Yusuke Iwasawa, and Yutaka Matsuo. Adopt: Modified adam can converge with any β_2 with the optimal rate. *arXiv preprint arXiv:2411.02853*, 2024.