Optimizer Inclusions

Ashwin Mathur Varun Mathur
{ashwinxmathur, varunm500}@gmail.com

September 23, 2024

Abstract

The choice of optimization algorithm for training Large Language Models (LLMs) significantly impacts both training speed and final predictive performance. In this paper, we demonstrate the critical importance of hyperparameter tuning protocols in optimizer comparisons for LLMs. Our research reveals that inclusion relationships between optimizers play a crucial role in practice and consistently predict optimizer performance. Contrary to conventional wisdom, we find that when carefully tuned, adaptive gradient methods such as Adam never underperform simpler optimizers like momentum or stochastic gradient descent (SGD). Our experiments show that more general optimizers consistently outperform their special cases, highlighting the practical significance of inclusion relationships between optimizers. We illustrate the sensitivity of optimizer comparisons to hyperparameter tuning protocols by examining previous experimental evaluations. Our results demonstrate how changes in tuning protocols can significantly alter optimizer rankings for a given workload (model and dataset pair). Notably, as more effort is invested in tuning, optimizer rankings stabilize according to their inclusion relationships. To validate our findings, we conducted extensive hyperparameter tuning across three NLP tasks: Sentiment Analysis, Question Answering, and Text Summarization. We utilized well-known foundational language models and fine-tuned them on various datasets for each task. For Sentiment Analysis, we used Financial Phrasebank, StockTwits, and FinGPT-Sentiment datasets. Question Answering experiments were conducted on SQuAD, CoQA, and FIQA datasets, while Summarization tasks employed Multi-News and BillSum datasets. Using these fine-tuned models, we demonstrate the inclusion relationships for a range of optimizers, including Adam, RMSProp, Nesterov Accelerated Gradient (NAG), SGD with momentum, and vanilla SGD. Our findings underscore that the differences between optimizers are entirely captured by their update rules and hyperparameters, with more expressive optimizers consistently outperforming their less expressive counterparts as hyperparameter tuning approaches optimality.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in various Natural Language Processing (NLP) tasks, including sentiment analysis, question answering, and text summarization. These tasks are particularly crucial in the financial domain, where vast amounts of information from reports and documents need to be processed to support critical decision-making involving significant financial stakes.

Sentiment analysis models can predict the emotional impact of events on company performance, question answering systems aid in financial analysis by providing quick responses to specific queries, and automatic summarization systems help reduce the time and effort required for financial decision-making. Given the complexity and importance of these tasks, optimizing the performance of LLMs is paramount.

A critical aspect of LLM optimization is the choice and tuning of the optimization algorithm used during training. This choice significantly impacts both the training speed and the final predictive performance of the model. Conventional wisdom in the field has often favored certain optimizers over others, but the true performance of these algorithms may be more nuanced than previously thought. In this paper, we challenge existing assumptions about optimizer performance by conducting a comprehensive study of hyperparameter tuning protocols and their impact on optimizer comparisons. Our research focuses on three key NLP tasks: Sentiment Analysis, Question Answering, and Text Summarization, using well-known foundational language models.

For Sentiment Analysis, we fine-tuned DistilBERT, BERT, FinBERT, Llama and Phi models. In the Question Answering task, we employed DistilBERT, BERT, RoBERTa, Llama and Phi models. Lastly, for Text Summarization, we utilized DistilBART, T5, and BART models. These models were fine-tuned on a variety of relevant datasets for each task, allowing us to explore optimizer performance across different model architectures and problem domains.

Our study examines the performance of several popular optimizers, including Adam, RMSProp, Nesterov Accelerated Gradient (NAG), SGD with momentum, and vanilla SGD. We investigate how the inclusion relationships between these optimizers affect their relative performance and how this performance varies with different hyperparameter tuning protocols.

By conducting this deep dive into hyperparameter tuning and optimizer comparisons, we aim to provide valuable insights that can guide researchers and practitioners in selecting and tuning optimizers for LLM training. Our findings have significant implications for improving the efficiency and effectiveness of LLM training across various NLP tasks, particularly in domains like finance where model performance can have substantial real-world impact.

2 Foundational Language Models

A language model is a probability distribution over words or word sequences. Language models interpret this data by feeding it through an algorithm that establishes rules for context in natural language. Then, the model applies these rules in language tasks to accurately predict or produce new sentences. The model essentially learns the features and characteristics of the basic language and uses those features to understand new phrases.

2.1 BERT

BERT (Devlin et al. 2018) makes use of the Transformer architecture, an attention mechanism that learns contextual relations between words (or sub-words) in a text. In the BERT architecture 15% of the tokens are masked while pretraining the model. Before feeding word sequences into BERT, 15% of the words in each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.

The model receives pairs of sentences as input and learns to predict if the second sentence in the pair is the subsequent sentence in the original document. During training, 50% of the inputs are a pair in which the second sentence is the subsequent sentence in the original document, while in the other 50% a random sentence from the corpus is chosen as the second sentence. The assumption is that the random sentence will be disconnected from the first sentence. When training the BERT model, Masked LM and Next Sentence Prediction are trained together, with the goal of minimizing the combined loss function of the two strategies.

Masked Language Modeling (MLM) involves randomly masking a percentage of tokens in a text, and using surrounding text to predict the masked tokens. The task of masking tokens in a sequence with a masking token and directing the model to fill that mask with an appropriate token is known as masked language modeling. This allows the model to focus on both the right and left contexts (tokens on the right side of the mask and tokens on the left of the mask). A variety of masking techniques have been used for domain specific pre-training. While some works propose rule based masking strategies that work better than random masking, other works attempt to find optimal masking policy automatically using techniques such as reinforcement learning. For masked language modelling, BERT based model takes a sentence as input and masks 15% of the words from a sentence and by running the sentence with masked words through the model, it predicts the asked words and context behind the words. BERT uses the [MASK] token 80% of the time, a random token 10% of the time, and the original token for the remaining 10% of the time to perform masking.

Many important downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI) are based on understanding the relationship between two sentences, which is not directly captured by language modeling. In order to train a model that understands sentence relationships, we pretrain for a binarized next sentence prediction task that can be trivially generated from any monolingual corpus. Specifically, when choosing the sentences A and B for each pretraining example, 50% of the time B is the actual next sentence that follows A (labeled as IsNext), and 50% of the time it is a random sentence from the corpus (labeled as NotNext).

2.2 DistilBERT

DistilBERT (Sanh et al. 2019) is a small, fast, cheap and light Transformer model trained by distilling BERT-base. It has 40% less parameters than bert-base, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark.

Knowledge distillation (Hinton, Vinyals, Dean, et al. 2015) is a prominent method for training compact networks to achieve comparable performance to a deep network. The concept of distillation is quite intuitive, it is the process of training a small student model to mimic a larger teacher model as close as possible. Distillation would be useless if we only run machine-learning models on the cluster we use to fine-tune them, but sadly, it isn't the case. Therefore, distillation comes in whenever we want to port a model onto smaller hardware, such as a limited laptop or a cellphone, because a distilled model runs faster and takes less space.

In the simplest form of distillation, knowledge is transferred to the distilled model by training it on a transfer set and using a soft target distribution for each case in the transfer set that is produced by using the teacher model. When the correct labels are known for all or some of the transfer set, this method can be significantly improved by also training the distilled model to produce the correct labels. This is done by using the correct labels to modify the soft targets. This can also be achieved by computing the objective function with the cross entropy of the soft targets and softmax values of the distilled model.

BERT has millions of parameters. Due to its large size, it is challenging to apply it in a real-world task. DistilBERT aims to reduce computation time by reducing the number of parameters by half. To compress the model size, DistilBERT applies the "teacher-student" framework also referred to as knowledge distillation where a larger model or the "teacher" network is trained and the knowledge is passed on to the smaller model also known as the "student" network. A smaller language model is trained by removing the token-type embeddings while reducing the number of parameters. This largely impacted the computation efficiency. According to GLUE standards, DistilBERT retains 97% performance of BERT with 40% fewer parameters and faster inference time. With the combination of transfer distillation and a two-stage learning framework, researchers reached the accuracy of general BERT with a model that is much smaller and faster.

2.3 FinBERT

FinBERT (Araci 2019; Yang, Uy, and A. Huang 2020) is a pre-trained NLP model to analyze the sentiment of financial text. It is built by training the BERT language model in the finance domain, using a large financial corpus and fine-tuning it for financial sentiment classification. FinBERT model is built on the standard BERT architecture. The model architecture is modified based on different downstream tasks such as sentiment analysis and question-answering by adding additional dense layers to the existing architecture.

The existing next sentence prediction loss is dropped and only the masked language modelling loss is used to update the weights. All the previous weights are frozen and only the weights between the final dense layers are updated. The model weights are initialized from the pretrained weights of the BERT model trained on the Wikipedia and News Corpus datasets. The model has been pre-trained on the Reuters TRC2 corpus. After that the model is fine-tuned by adding a dense layer for the sentiment analysis dataset. Although the second corpus is much smaller, using data from the direct target provides better target domain adaptation.

2.4 RoBERTa

RoBERTA (Y. Liu et al. 2019) is a modified version of BERT. It removes the next sentence prediction objective and uses dynamic masking during pretraining. It performs better than BERT as it is trained with a larger batch size and more data. RoBERTa has a similar architecture as compared to BERT, but in order to improve the results on BERT architecture, there are some changes in its architecture and training procedure. These changes are:

- Removing the Next Sentence Prediction (NSP) objective: In next sentence prediction, the model is trained to predict whether the observed document segments come from the same or distinct documents via an auxiliary Next Sentence Prediction (NSP) loss. The authors experimented with removing/adding of NSP loss to different versions and concluded that removing the NSP loss matches or slightly improves downstream task performance.
- Training with bigger batch sizes and longer sequences: Originally BERT is trained for 1M steps with a batch size of 256 sequences. In this paper, the authors trained the model with 125 steps of 2K sequences and 31K steps with 8k sequences of batch size. This has two advantages. First, the large batches improve perplexity on the masked language modelling objective as well as end-task accuracy. Second, large batches are also easier to parallelize via distributed parallel training.
- Dynamically changing the masking pattern: In the BERT architecture, the masking is performed once during data preprocessing, resulting in a single static mask. To avoid using the single static mask, training data is duplicated and masked 10 times, each time with a different mask strategy over 40 epochs thus having 4 epochs with the same mask. This strategy is compared with dynamic masking in which different masking is generated each time we pass data into the model.

2.5 T5

T5 (Raffel et al. 2020) is an encoder-decoder model pre-trained on a multi-task mixture of unsupervised and supervised tasks and for which each task is converted into a text-to-text format. In contrast to other Transformer based models like BERT or GPT, which are made up of either the encoder or decoder part of the Transformer, T5 uses the complete encoder-decoder architecture and shows that this is better than only using the decoder. The model was pre-trained on the Colossal Clean Crawled Corpus (C4), which is a huge crawled and cleaned dataset using self-supervised learning. Due to this nature of T5, training or fine-tuning the model requires a pair of input and output sequences/text.

T5 is a combination of three different architectures:

- Encoder-Decoder: A standard encoder-decoder architecture uses fully visible masking in the encoder and the encoder-decoder attention, with causal masking (attention mask) in the decoder. Masking is done to make sure the output at a position doesn't attend to future output for prediction.
- Language model: A language model consists of a single Transformer layer stack and is fed the concatenation of the input and target, using a causal mask throughout. The output only attends to the past input or output.
- Prefix based Language Modeling: Adding a prefix to a language model corresponds to allowing fully-visible masking over a portion of the input. It is very similar to language models, just that any output will attend to a certain portion of the input which contains a prefix with task specific information like translate English to German.

2.6 BART

BART (Lewis et al. 2019) is a transformer encoder-encoder model with a bidirectional BERT-like encoder, and an autoregressive GPT-like decoder. BART is pre-trained by corrupting text with an arbitrary noising function, and learning a model to reconstruct the original text. BART uses the basic sequence-to-sequence Transformer architecture with the difference that the activation functions for ReLUs are changed to GeLUs, and the initialization parameters are changed as per the GPT decoder.

The architecture is similar to BERT's, although there are a few differences. As compared to the Transformer architecture, each decoder layer in the model performs cross-attention across the encoder's final hidden layer. Another significant difference is that BERT utilises an additional feed-forward network before word prediction, whereas BART does not. BART has around 10% more parameters than a BERT model of comparable size. BART is trained by corrupting documents and then increasing the reconstruction loss, which is the difference in cross-entropy between the decoder output and the original document.

The corruption schemes that were used in the pre-training of BART are:

- Token Masking : A random subset of the input is replaced with [MASK] tokens, like in BERT.
- Token Deletion: Random tokens are deleted from the input. The model must decide which positions are missing (as the tokens are simply deleted and not replaced with anything else).
- Text Infilling: A number of text spans (length can vary) are each replaced with a single [MASK] token.
- Sentence Permutation: The input is split based on periods (.), and the sentences are shuffled.
- Document Rotation: A token is chosen at random, and the sequence is rotated so that it starts with the chosen token.

2.7 Phi-3-mini-128k-instruct

The Phi-3-Mini-128K-Instruct (Abdin et al. 2024) is a 3.8 billion-parameter, lightweight, state-of-the-art open model trained using the Phi-3 datasets. When assessed against benchmarks testing common sense, language understanding, math, code, long context and logical reasoning, Phi-3 Mini-4K-Instruct showcased a robust and state-of-the-art performance among models with less than 13 billion parameters. Phi-3-mini-128k-instruct has a maximum context length of 131072 tokens.

2.8 Llama-3-8B

Llama-3-8B (Touvron et al. 2023) is an auto-regressive language model that uses an optimized transformer architecture. It has been trained using supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF) to align with human preferences for helpfulness and safety.

The model uses a tokenizer with a vocabulary of 128K tokens that encodes language much more efficiently, which leads to substantially improved model performance. The model was trained on sequences of 8,192 tokens, using a mask to ensure self-attention does not cross document boundaries and also uses grouped query attention. It has a maximum context length of 8192 tokens.

2.9 HuggingFace Pre-trained Models

In our experiments described below, we used the models from Huggingface's Transformers library (Wolf et al. 2019). Each of the Transfomer models was initialized with their respective pretrained version. The links to the pretrained model checkpoints for each model can be found in Table 1.

Model	**HuggingFace Model**
BERT	bert-base-uncased
DistilBERT	distilbert-base-uncased
FinBERT	finbert
RoBERTa	roberta-base
T5	t5-base
BART	bart-base
Phi-3-mini-128k-instruct	Phi-3-mini-128k-instruct
Llama-3-8B	Meta-Llama-3-8B

Table 1: HuggingFace Models

2.10 Model Training Procedure using HuggingFace

The first step in the training process of the HuggingFace model is to tokenize the input sequences. The input sequences to the model are tokenized based on the type of the model. Each input sequence is passed through a Token Embedding layer and transformed into a vector representation. These representations are in the form of Segment Embeddings and Position Embeddings. Based on the architecture, the model configuration parameters are initialized. Parameters such as the vocabulary size, the number of hidden layers, and the number of attention heads control the size and complexity of the model. The weights of the model are initialized from the pretrained checkpoints available on HuggingFace Hub. These pretrained checkpoints of the model have been trained using different datasets and text corpus.

An extra output head is added on top of the existing architecture so that the model can be used for downstream tasks like Sentiment Analysis, Question-Answering and Text Summarization. This is done by adding a series of Dense layers to the model based on the task to be performed. The number of Dense layers that are required for the output head are determined by the type of downstream task. For the Sentiment Analysis task, a single dense layer is added. For the Question-Answering task, two output heads are usually added, as they are reuired for predicting the start and end positions of the generated answer. For the Text Summarization task usually a decoder transformer block is initialized. The weights for the final output heads are trained by the method of layer freezing. All the previous trained weights are frozen and only the weights between the final output head are updated.

3 Analysis of Optimizers and Hyperparameter Tuning

The choice of the optimizer and the associated hyperparameters in model training can hugely impact the performance of the model. In the past, empirical comparison of optimizers (Choi et al. 2019) has been done to explore the effect of different optimizers on convergence of training for different tasks. Selecting an optimizer is a central step in training deep learning learning models. The paper demonstrates the sensitivity of optimizer comparisons to the hyperparameter tuning protocol. Their findings suggest that the hyperparameter search space may be the single most important factor explaining the rankings obtained by recent empirical comparisons in the literature. The paper demonstrates two important and interrelated points about empirical comparisons of neural network optimizers. First, it shows that inclusion relationships between optimizers actually matter in practise. More general optimizers never underperform special cases. Second, it demonstrates the sensitivity of optimizer comparisons to the hyperparameter tuning protocol. They perform a comparison of optimizers for Image Classification on the Fashion MNIST, CIFAR-10, CIFAR-100 and ImageNet datasets. They also perform comparisons for language modeling on War and Peace and LM1B datasets.

3.1 Importance of Hyperparameter Tuning

Optimization algorithms are defined by their update rule, which is controlled by hyperparameters that determine its behaviour. The difference between optimizers is entirely captured by the choice of update rule and hyperparameters. Selecting the hyperparameter search space for each optimizer is a key methodological choice for any empirical comparison of optimizers.

In our experiments, we chose the search space for each optimizer by running an initial set of experiments over a relatively large search space. We ran a single set of initial trials per optimizer to select the final search space. We evaluated the performance of different optimizers when training models for tasks like Sentiment Analysis, Question Answering and Text Summarization. The comparison of convergence of optimizers was tested for the best performing models on all the datasets. Five different optimizers namely AdamW, RMSProp, Nesterov Accelerated Gradient Descent(NAG), Stochastic Gradient Descent with Momentum and Stochastic Gradient Descent were compared to measure the convergence of the model on the dataset.

3.2 Optimizer Hyperparameters

We considered the following optimizer hyperparameters:

- **Learning Rate (η):**
  The Learning Rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging since a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.
- **Momentum (γ):**
  The Momentum hyperparameter allows the gradient search to build inertia in a direction in the search space, and overcome the oscillations of noisy gradients and converge to the local maxima/minima faster. Momentum has the effect of dampening down the change in the gradient and the step size with each new point in the search space.
- **Smoothing Constant (α):**
  The Smoothing Constant hyperparameter maintains a decaying average of squared gradients. Using a decaying moving average of the partial derivative allows the search to forget early partial derivative values and focus on the most recently seen shape of the search space. This helps the optimizer to discard history from the extreme past so that it can converge rapidly after finding a convex surface.
- **Exponential Decay Rate - First Order and Second Order Moments (β_1 and $\beta 2$):**
  The exponential decay rate for the first order moment estimates, and the second order moment estimates, are used for smoothing the path to convergence, and for also providing some momentum to cross a local minima or saddle point. These are similar to the smoothing constant in the other optimizers, but are used to decay the gradients instead of the learning rate.

3.3 Stochastic Gradient Descent (SGD) Optimizer

Stochastic Gradient Descent (SGD) is an optimization algorithm that estimates the error gradient for the current state of the model using training examples, and then updates the weights of the model using backpropagation. The amount by which the weights are updated during training is affected by the learning rate. The learning rate is the only tunable hyperparameter for SGD. The choice of the learning rate is crucial as choosing a learning rate which is too low will cause the model to converge slowly, whereas a learning rate which is too high may cause it to not converge at all.

3.4 Stochastic Gradient Descent with Momentum Optimizer

Stochastic Gradient Descent with Momentum is an optimization algorithm which uses momentum to accelerate the gradients in the right directions, thus leading to faster convergence. Momentum allows the search to build inertia in a direction in the search space, and overcomes the oscillations of noisy gradients and converges to the local maxima/minima faster. The tunable hyperparameters for SGD with momentum are the learning rate and momentum. The momentum term increases for dimensions whose gradients point in the same directions, and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

3.5 Nesterov Accelerated Gradient Descent(NAG) Optimizer

Nesterov Accelerated Gradient Descent (NAG) is a modification of SGD with momentum. NAG follows a gradient look-ahead approach to decide the direction of the gradient. The Nesterov accelerated momentum helps fix the problem of overshooting the minima/maxima. NAG first makes a big jump in the direction of the previous accumulated gradient, measures the gradient, and then makes a correction. This anticipatory update prevents us from going too fast and results in increased responsiveness and faster convergence. The tunable parameters for NAG are learning rate and momentum. The update in NAG happens in two steps. First, a partial step is taken to reach the look ahead point, and then the final update is made. We calculate the gradient at the look-ahead point and then use it to calculate the final update.

3.6 RMSProp Optimizer

RMSProp is a modification of SGD with momentum. RMSProp uses an adaptive learning rate to reach convergence quickly. It tries to resolve the problem that gradients may vary widely in magnitudes. The algorithm uses momentum just like SGD to accelerate the search for the mimima/maxima. It combines the idea of only using the sign of the gradient with the idea of adapting the step size individually for each weight. The smoothing constant stores the sum of moving average of squared gradients for every weight. The tunable parameters for RMSProp are learning rate, momentum, and the smoothing constant. The key difference is that the smoothing constant controls how far to move in the search space against the gradient for each iteration of the algorithm.

3.7 AdamW Optimizer

AdamW uses the first and second order moments to estimate the learning rate decay for the adaptive learning rate. Unlike maintaining a single learning rate through training in SGD, AdamW optimizer updates the learning rate for each network weight individually. The tunable parameters for AdamW are learning rate, momentum, alpha, beta1 and beta2. AdamW takes into consideration the exponentially weighted average of the gradients. The β_1 parameter is used to set the exponential decay rate for the first moment estimate, and the β_2 parameter is the exponential decay rate for the second moment estimate.

3.8 Gradient Clipping

- Gradient clipping is a technique that tackles exploding gradients. The idea of gradient clipping is very simple: If the gradient gets too large, rescale it to keep it small.
- In Gradient clipping by value, if a gradient exceeds a threshold value, the gradient is clipped to the threshold. If the gradient is less than the lower limit then it is is clipped too, to the lower limit of the threshold.
- In Norm gradient clipping, the gradients are clipped by multiplying the unit vector of the gradients with the threshold.
- This ensures that the global norm of all the gradients calculated is not more than 1.

3.9 Learning Rate Scheduler

- The learning rate governs the magnitude of the step taken by the optimizer in its pursuit of minimizing the loss function.
- A learning rate schedule is a predefined framework that adjusts the learning rate between epochs or iterations as the training process progresses.
- During the initial stages of training, the learning rate is set to a higher value to facilitate the attainment of a set of weights that are reasonably accurate. As training continues, these weights are fine-tuned through the application of a smaller learning rate, thereby achieving higher accuracy.
- Learning rate schedules aim to adjust the learning rate during the training process by reducing the learning rate according to a predefined schedule. Commonly employed learning rate schedules include time-based decay, step decay, and exponential decay.
- This approach enables the model to make substantial updates at the beginning of training when the parameters are far from their optimal values, and smaller updates later when the parameters are closer to their optimal values, thereby allowing for faster convergence.

3.10 Gradient Accumulation

- During the training of neural networks, data is typically divided into mini-batches, which are processed iteratively. For each mini-batch, the network computes predicted labels, and the loss is calculated with respect to the ground truth targets. Subsequently, a backward pass is performed to compute the gradients, facilitating the updating of model weights in the direction of the gradients.
- Gradient accumulation modifies the final step of the training process. Instead of updating the network weights after processing each mini-batch, the gradient values are accumulated. The model proceeds to the next mini-batch, and the new gradients are added to the accumulated gradients. The weight update is then performed after several mini-batches have been processed by the model.
- The primary advantage of gradient accumulation is that it effectively simulates a larger batch size during training.
- Gradient accumulation provides a means to virtually increase the batch size during training, which is particularly beneficial when the available GPU memory is insufficient to accommodate the desired batch size.
- In the gradient accumulation technique, gradients are computed for smaller mini-batches and accumulated (typically through summation or averaging) over multiple iterations, rather than updating the model weights after each mini-batch. Once the accumulated gradients reach the target "virtual" batch size, the model weights are updated using the accumulated gradients.

3.11 Hyperparameters used for Tuning Optimizers

The performance of a model significantly depends on the value of hyperparameters. Grid Search is the process of performing hyperparameter tuning in order to determine the optimal values for a given model. This function helps to loop through predefined hyperparameters and fits the estimator (model) on the training set. So, in the end, one can select the best parameters from the list of hyperparameters. For fine tuning the parameters of each optimizer, we have taken a grid search with a step size of 10 over a large search space. The search space is reduced, and the optimal value of the hyperparameter is selected for training the model.

3.12 Hyperparameter Tuning Search Space

For each optimizer, the initial and final search spaces have been mentioned. For fine tuning the optimizer, an initial search space for the parameters is used. A smaller search space is used to find the optimal value of the parameter.

	η (Learning Rate)
Initial	[1e-7, 1e-3]
Final	[1e-5, 1e-1]

Table 2: SGD Hyperparameter Search Space

	η (Learning Rate)	γ (Momentum)
Initial	[1e-4, 1e-1]	[1e-4, 1]
Final	[1e-3, 1e-1]	[1e-3, 1]

Table 3: SGD (Momentum) Hyperparameter Search Space

	η (Learning Rate)	γ (Momentum)
Initial	[1e-4, 1e-1]	[1e-4, 1]
Final	[1e-3, 1e-1]	[1e-3, 1]

Table 4: NAG Search Space

	η (Learning Rate)	γ (Momentum)	α (Alpha)	ϵ (Epsilon)
Initial	[1e-4, 1e-1]	[1e-4, 1]	[0, 1-1e-2]	[1e-9, 1e-5]
Final	[1e-5, 1]	[1e-3, 1e-1]	[0, 1-1e-1]	[1e-7, 1e-5]

Table 5: RMSProp Search Space

	η (Learning Rate)	γ (Momentum)	α (Alpha)	β1 (Beta1)	β2 (Beta2)	ϵ (Epsilon)
Initial	[1e-4, 1e-1]	[1e-4, 1]	[0, 1-1e-2]	[1-1e-1, 1-1e-5]	[1-1e-1, 1-1e-3]	[1e-2, 1e-5]
Final	[1e-3, 1e-1]	[1e-3, 1]	[1-1e-2, 1-1e-4]	[1-1e-1, 1-1e-2]	[1-1e-1, 1-1e-2]	[1e-5, 1e-7]

Table 6: AdamW Search Space

4 Sentiment Analysis

Sentiment analysis is the task of extracting sentiments or opinions of people from written language.

The input sequence to the model was converted to token and postion embeddings. The [CLS] and [SEP] tokens were added to the beginning and end of the sequence respectively. A dense layer was added after the last hidden state of the [CLS] token to perform the sentiment classification. The models were trained on the labeled datasets to learn the token embeddings and predict the correct sentiment.

4.1 Datasets

The Financial PhraseBank(Malo et al. 2014) and StockTwits (Jaggi et al. 2021) datasets were used for the Sentiment Analysis task. The Sentiment Analysis model predicts the sentiment or polarity of the given financial headline or stocktwit.

4.1.1 Financial PhraseBank

The Financial PhraseBank dataset consists of 4,840 sentences from English language financial news categorized by sentiment. These sentences then were annotated by 16 people with a background in finance and business.

The dataset consists of:

- **Sentiment:** The sentiment can be negative, neutral or positive.
- **News Headline:** Headlines of the news articles.

4.1.2 StockTwits

The StockTwits dataset consists of 6,451,067 StockTwits from the StockTwits platform. Each post has been categorized by a polarity.

The dataset consists of:

- **Text:** The text contained in the StockTwit post. The text refers to information about the performance of a stock.
- **Polarity:** Contains two values ie. Negative and Positive. Negative indicates that the Stock price may fall, whereas Positive indicates an increase in price of the Stock.

4.1.3 FinGPT-Sentiment

The FinGPT-Sentiment dataset is a comprehensive sentiment analysis corpus that incorporates explicit instructions for sentiment classification tasks. This dataset is a composite of several existing financial sentiment datasets, including the Financial Phrase Bank, FIQA, Twitter-Financial-News-Sentiment, and News-With-GPT-Instructions. The dataset is structured to include specific sentiment analysis prompts alongside the corresponding sentiment scores. A typical instruction in the dataset reads: "What is the sentiment of this news? Please choose an answer from (strong negative/moderately negative/neutral/moderately positive/strong positive)." This format ensures consistency in sentiment classification and provides clear guidelines for model training and evaluation. The FinGPT-Sentiment dataset comprises approximately 76,800 training samples, offering a substantial corpus for developing and fine-tuning sentiment analysis models in the financial domain.

The dataset consists of:

- **Input:** This field contains the textual content extracted from the headlines of news articles. These headlines serve as the primary text for sentiment analysis.
- **Instruction:** This column provides explicit directives or prompts that guide the sentiment analysis task. These instructions are designed to direct the model in evaluating the sentiment of the text provided in the input column. Furthermore, they delineate the specific sentiment categories from which the model should select its classification.
- **Output:** This field encompasses the sentiment labels assigned to each input text. The labels span a range of sentiment categories, including:
  - Broad categories: positive, negative, and neutral
  - Fine-grained categories: mildly positive, mildly negative, moderately positive, and moderately negative

  This granular classification scheme allows for a more nuanced analysis of sentiment in financial news headlines.

4.2 Evaluation Method

The performance of the models were evaluated using the following metrics:

- **Accuracy:** Proportion of correct predictions with respect to Total number of predictions.
- **F1 Score(weighted):** Weighted Sum of per class F1 Scores.

4.3 Fine-tuning the models

For the Financial PhraseBank dataset, a baseline was set using the BERT model. The BERT model was fine-tuned using an AdamW optimizer with learning rate of 5e-5, and a batch-size of 32. The FinBERT and DistilBERT models were also fine-tuned using an AdamW optimizer with learning rate of 5e-5 and a batch-size of 32. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of the Accuracy and F1-Score(Weighted) on the test set. The FinBERT model achieved the best performance on the Finanacial PhraseBank dataset. Table 7 lists out the performance of the different BERT architectures on the Financial PhraseBank dataset.

Model	Accuracy	F1 Score(Weighted)
DistilBERT	0.82	0.81
BERT	0.84	0.82
FinBERT	0.85	0.83

Table 7: Experiment results on the Financial PhraseBank Dataset

For the StockTwits dataset, a baseline was set using the BERT model. The BERT model was fine-tuned using an AdamW optimizer with learning rate of 5e-5, and a batch-size of 32. The DistilBERT and FinBERT models were fine-tuned using an AdamW optimizer with learning rate of 3e-5 and a batch-size of 32. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of the Accuracy and F1-Score(Weighted) on the test set. The FinBERT model achieved the best performance on the StockTwits dataset. Table 8 lists out the performance of the different BERT architectures on the StockTwits dataset.

Model	Accuracy	F1 Score(Weighted)
DistilBERT	0.64	0.38
BERT	0.74	0.68
FinBERT	0.82	0.68

Table 8: Experiment results on the StockTwits Dataset

The Llama-3 model was fine-tuned using an AdamW optimizer with learning rate of 3e-5, and a batch-size of 16 on the FinGPT-Sentiment dataset. The Phi-3 model was also fine-tuned using an AdamW optimizer with learning rate of 3e-5 and a batch-size of 16. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of F1-Score(Weighted) on the test set. The Llama-3 model achieved the best performance on the FinGPT-Sentiment dataset. Table 9 lists out the performance of the different LLMs architectures on the FinGPT-Sentiment dataset.

Model	Accuracy	F1 Score(Weighted)
Llama-3	0.84	0.68
Phi-3	0.74	0.59

Table 9: Experiment results on the FinGPT-Sentiment Dataset

4.4 Optimizer Analysis

The optimizers were tuned for different parameters independently by a way of specific search spaces. Each parameter was first tuned on a large search space, and then a smaller and more refined search space was used to find the optimal value for training the model. The tuned hyperparameters were used to generate the plots for comparison of the optimizers.

As shown in Table 2, the Learning Rate was tuned over an initial search space ranging from 1e-7 to 1e-3. The final search space was found to be 5e-6 to 1e-4. The optimal value of Learning Rate was found to be 5e-5 for the Financial Phrasebank and StockTwits datasets. The optimal value of learning rate was found to be 3e-5 for the FinGPT-Sentiment dataset.

As shown in Table 3, the Momentum was tuned over an initial search space ranging from 1e-4 to 1e-1. The final search space was found to be 1e-3 to 1. The optimal value of Momentum was found to be 1e-3 for both the datasets.

Optimizer	η **(Learning Rate)**	γ **(Momentum)**	α **(Alpha)**	β_1 **(Beta1)**	β_2 **(Beta2)**	ϵ **(Epsilon)**
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	3e-5	1e-1	1-1e-1	-	-	1e-5
AdamW	5e-5	1e-1	1-1e-1	1-1e-1	1-1e-1	1e-5

Table 10: Optimizer Hyperparameters for fine-tuning FinBERT on Financial PhraseBank.

As shown in Table 4, the Nesterov Momentum was tuned over an initial search space ranging from 1e-4 to 1.The final search space was found to be 1e-3 to 1e-1. The optimal value of Nesterov Momentum was found to be 1e-3 for both the three datasets.

As shown in Table 5, the smoothing constant was tuned over an initial search space ranging from 0 to 1-1e-2.The final search space was found to be 0 to 1-1e-1. The optimal value of the smoothing constant was found to be 1-1e-1 for both the datasets.

Optimizer	η **(Learning Rate)**	γ **(Momentum)**	α **(Alpha)**	β_1 **(Beta1)**	β_2 **(Beta2)**	ϵ **(Epsilon)**
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	2e-5	1e-1	1-1e-2	-	-	1e-5
AdamW	3e-5	1e-1	1-1e-2	1-1e-2	1-1e-2	1e-5

Table 11: Optimizer Hyperparameters for fine-tuning FinBERT on StockTwits Datasets.

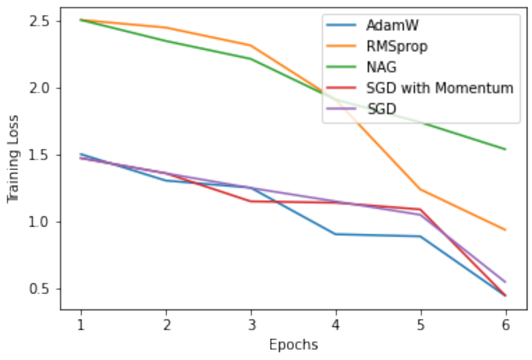
As shown in Table 6, the exponential decay rate of the first and second moments were tuned over an initial search space ranging from 1-1e-1 to 1-1e-5, and 1-1e-1 to 1-1e-2. The optimal value of β_1 was found to be 1-1e-2, and the optimal value of β_2 (beta2) was found to be 1-1e-2 for both the datasets. The final values of the hyperparameters are shown in Table 10 and Table 11.

Optimizer	**Learning Rate** (η)	**Momentum** (γ)	**Alpha** (α)	**Beta1** (β_1)	**Beta2** (β_2)	**Epsilon** (ϵ)
SGD	0.01	-	-	-	-	-
SGD (Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	2e-5	1e-1	0.99	-	-	1e-5
AdamW	3e-5	1e-1	0.99	0.99	0.99	1e-5

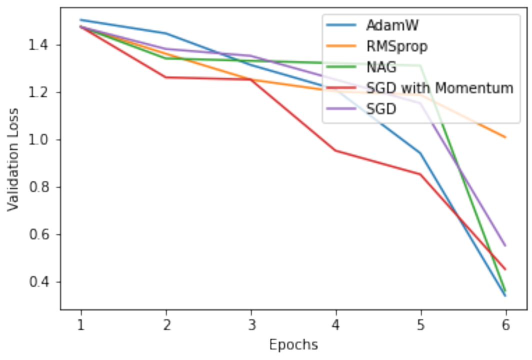
Table 12: Optimizer Hyperparameters for fine-tuning Llama-3 on FinGPT-Sentiment dataset

As shown in Table 6, the exponential decay rate of the first and second moments were tuned over an initial search space ranging from 1-1e-1 to 1-1e-5, and 1-1e-1 to 1-1e-2. The optimal value of β_1

was found to be 1-1e-2, and the optimal value of β_2 (beta2) was found to be 1-1e-2 for both the datasets. The final values of the hyperparameters are shown in Table 12.



(a) Training Loss

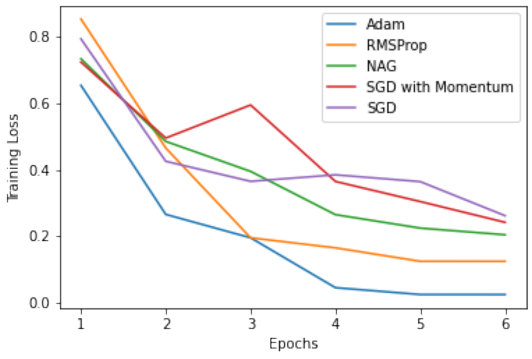


(b) Validation Loss

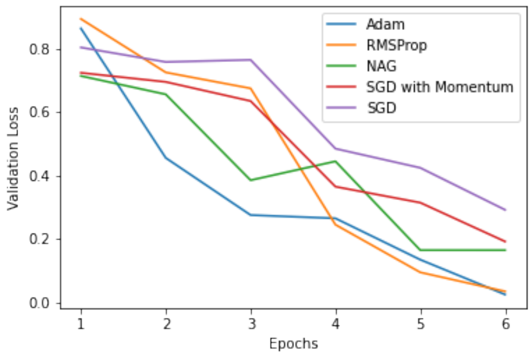
Figure 1: Comparison of the Optimizers on the Financial PhraseBank Dataset for the FinBERT model

The comparison of training and validation loss for all the optimizers is shown in Figure 1 and Figure 2.

For the Financial PhraseBank Dataset, the AdamW and RMSProp optimizers converge the quickest. It can be seen that the NAG optimizer gets stuck in a local minima and takes much longer to converge. The SGD and SGD with momentum optimizers are not able to converge in the same epochs as the others. The lowest loss was achieved by the AdamW optimizer.



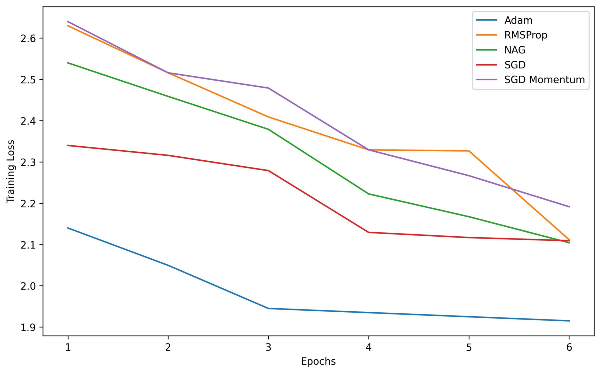
(a) Training Loss



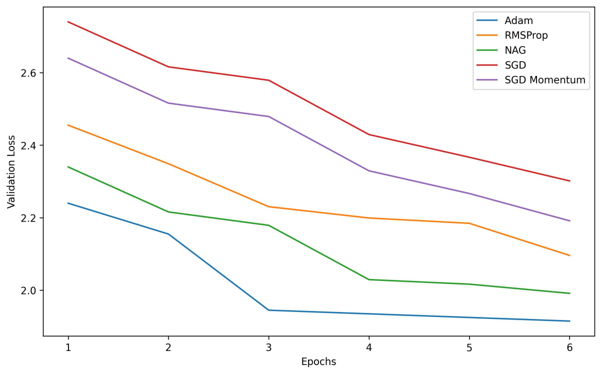
(b) Validation Loss

Figure 2: Comparison of the Optimizers on the StockTwits Dataset for the FinBERT model

For the Stocktwits Dataset, the AdamW and RMSProp optimizers converge the quickest. The SGD optimizer performs the worst and is not able to converge. We observe a spike in the loss of the NAG optimizer after four epochs indicating that it got stuck in local minima.



(a) Training Loss



(b) Validation Loss

Figure 3: Comparison of the Optimizers on the FinGPT-Sentiment dataset for the Llama-3 model

For the FinGPT-Sentiment Dataset, the AdamW and RMSProp optimizers converge the quickest. The SGD optimizer performs the worst and is not able to converge. We observe a spike in the loss of the NAG optimizer after four epochs indicating that it got stuck in local minima.

5 Question Answering

Question Answering models retrieve the answer to a question from a given text. Question answering models take two inputs, the input question, and the context passage. The answer sequence is present in the context passage. Given a question in natural language, the model extracts the relevant information from the provided data and presents it in the form of a natural language answer. This is done by predicting the starting and ending positions of the answer in the context passage.

The two input sequences to the model are converted into token embeddings and segment embeddings. The segment embeddings are created by adding a marker to the first token in the sentence. This allows the model to distinguish between sentences. The model uses segment embeddings to differentiate the question from the reference text. A [CLS] token is added to the input word tokens at the beginning of the question, and a [SEP] token is inserted at the end of both the question and the context. The embeddings are passed through the transformer layers and the weights are updated. A output head consisting of a pair of dense layers is used to generate the start and end positions of the answer in the context passage. This is then presented as the generated answer.

5.1 Datasets

The Stanford Question Answering Dataset (SQuAD) (Rajpurkar, Jia, and Liang 2018; Rajpurkar, Zhang, et al. 2016) and Conversational Question Answering Dataset (CoQA) (Reddy, D. Chen, and Manning 2019) were used for the Question Answering task. The Question Answering model predicts the positions of the answer text from the supplied context.

5.1.1 SQuAD

SQuAD is a reading comprehension dataset consisting of questions posed by crowdworkers on a set of Wikipedia articles. There are 100,000+ question-answer pairs on 500+ articles.

The dataset consists of:

- **Title:** The Title for each question and answer pair.
- **Context:** The context of the news article.
- **Question:** The question regarding the context.
- **Answers:** The answer to each question.

5.1.2 CoQA

CoQA contains 127,000+ questions with answers collected from 8,000+ conversations.The passages are collected from seven diverse domains.

The dataset consists of:

- **Text**: The Title for each question and answer pair

- **Question**: The question regarding the context.
- **Answer**: The answer to each question.

5.1.3 FiQA

The FIQA (Financial Opinion Mining and Question Answering) dataset has a corpus, queries and qrels (relevance judgments file). The FiQA dataset has roughly 6,000 questions and 57,000 answers. They are in the following format:

- **Corpus file**: a .jsonl file (jsonlines) that contains a list of dictionaries, each with three fields _id with unique document identifier, title with document title (optional) and text with document paragraph or passage.
- **Queries file**: a .jsonl file (jsonlines) that contains a list of dictionaries, each with two fields _id with unique query identifier and text with query text.
- **Qrels file**: a .tsv file (tab-seperated) that contains three columns, i.e. the query-id, corpus-id and score in this order.

5.2 Fine-tuning the models

For the SQuAD dataset, a baseline was set using the BERT model. The BERT model was fine-tuned using an AdamW optimizer with learning rate of 5e-5, and a batch-size of 32. The DistilBERT and RoBERTa models were also fine-tuned using an AdamW optimizer with learning rate of 5e-5 and a batch-size of 32. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of F1-Score(Weighted) on the test set. The RoBERTa model achieved the best performance on the SQuAD dataset. Table 13 lists out the performance of the different BERT architectures on the SQuAD dataset.

Model	F1 Score(Weighted)
DistilBERT	0.62
BERT	0.64
RoBERTa	0.74

Table 13: Experiment results on the SQuAD Dataset

For the CoQA dataset, a baseline was set using the BERT model. The BERT model was fine-tuned using an AdamW optimizer with learning rate of 5e-5, and a batch-size of 16. The DistilBERT and RoBERTa models were also fine-tuned using an AdamW optimizer with learning rate of 3e-5 and a batch-size of 16. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of F1-Score(Weighted) on the test set. The RoBERTa model achieved the best performance on the CoQA dataset. Table **??** lists out the performance of the different BERT architectures on the CoQA dataset.

Model	F1 Score(Weighted)
DistilBERT	0.68
BERT	0.72
RoBERTa	0.84

Table 14: Experiment results on the CoQA Dataset

The Llama-3 model was fine-tuned using an AdamW optimizer with learning rate of 3e-5, and a batch-size of 16 on the FIQA dataset. The Phi-3 model was also fine-tuned using an AdamW optimizer with learning rate of 3e-5 and a batch-size of 16. The models were trained for 6 epochs each. The performance of the model was evaluated on the basis of F1-Score(Weighted) on the test set. The Llama-3 model achieved the best performance on the FIQA dataset. Table 14 lists out the performance of the different LLM architectures on the FIQA dataset.

Model	F1 Score(Weighted)
LLama-3	0.86
Phi-3	0.75

Table 15: Experiment results on the FIQA Dataset

5.3 Optimizer Analysis

The optimizers were tuned for different parameters independently by way of specific search spaces. Each parameter was first tuned on a large search space and then a smaller and more refined search space was used to find the optimal value for training the model.

As shown in Table 2, the Learning Rate was tuned over an initial search space ranging from 1e-7 to 1e-3. The final search space was found to be 5e-6 to 1e-4. The optimal value of Learning Rate was found to be 5e-5 for the SQuAD dataset, 3e-5 for the CoQA dataset and 2e-5 for the FIQA dataset.

As shown in Table 3, the Momentum was tuned over an initial search space ranging from 1e-4 to 1e-1. The final search space was found to be 1e-3 to 1. The optimal value of Momentum was found to be 1e-3 for the three datasets.

Optimizer	η (Learning Rate)	γ (Momentum)	α (Alpha)	β_1 (Beta1)	β_2 (Beta2)	ϵ (Epsilon)
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	3e-5	1e-1	1-1e-2	-	-	1e-5
AdamW	5e-5	1e-1	1-1e-2	1-1e-2	1-1e-2	1e-5

Table 16: Optimizer Hyperparameters for fine-tuning RoBERTa on SQuAD dataset.

As shown in Table 4, the Nesterov Momentum was tuned over an initial search space ranging from 1e-4 to 1. The final search space was found to be 1e-3 to 1e-1. The optimal value of Nesterov Momentum was found to be 1e-3 for both the datasets. As shown in Table 5, the smoothing constant was tuned over an initial search space ranging from 0 to 1-1e-2. The final search space was found to be 0 to 1-1e-1. The optimal value of the smoothing constant was found to be 1-1e-2 for the three datasets.

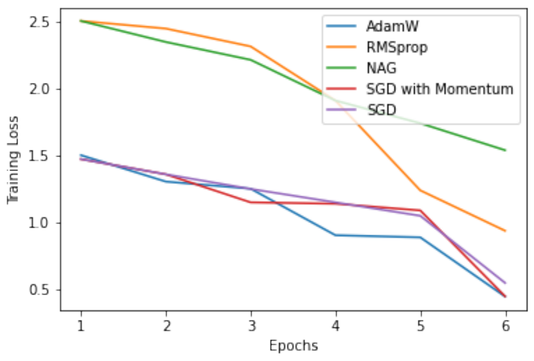
Optimizer	η (Learning Rate)	γ (Momentum)	α (Alpha)	β_1 (Beta1)	β_2 (Beta2)	ϵ (Epsilon)
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	2e-5	1e-1	1-1e-2	-	-	1e-5
AdamW	3e-5	1e-1	1-1e-2	1-1e-2	1-1e-2	1e-5

Table 17: Optimizer Hyperparameters for fine-tuning RoBERTa on CoQA Dataset.

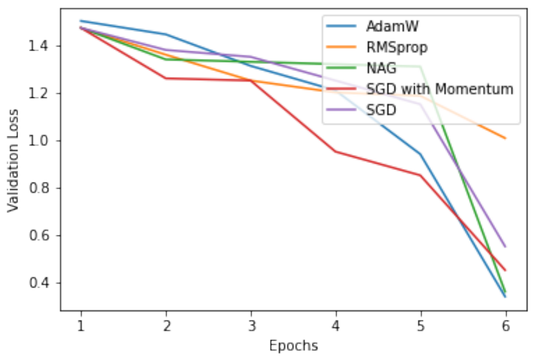
As shown in Table 6, the exponential decay rate of the first and second moments were tuned over an initial search space ranging from 1-1e-1 to 1-1e-5 and 1-1e-1 to 1-1e-2. The optimal value of β_1 was found to be 1-1e-2, and the optimal value of β_2 (beta2) was found to be 1-1e-2 for both the datasets. The final values of the hyperparameters are shown in Table 10 and Table 11.

Optimizer	**Learning Rate (η)**	**Momentum (γ)**	**Alpha (α)**	**Beta1 (β_1)**	**Beta2 (β_2)**	**Epsilon (ϵ)**
SGD	0.01	-	-	-	-	-
SGD (Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	3e-5	1e-1	0.99	-	-	1e-5
AdamW	2e-5	1e-1	0.99	0.99	0.99	1.e-5

Table 18: Optimizer Hyperparameters for fine-tuning Llama-3 on FiQA Dataset.



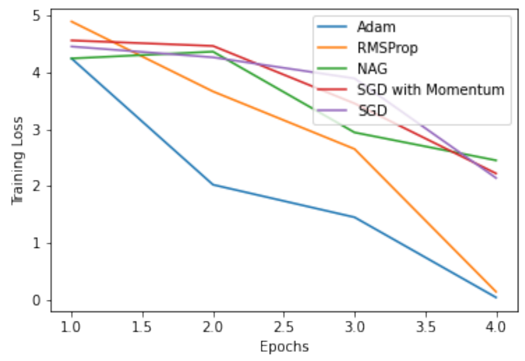
(a) Training Loss



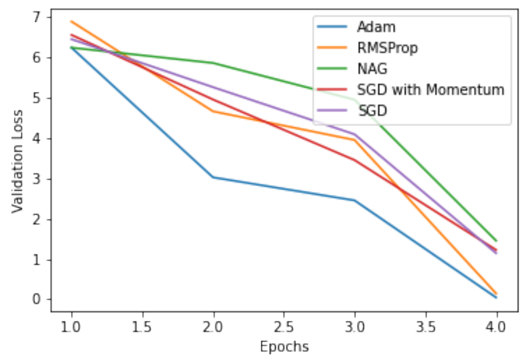
(b) Validation Loss

Figure 4: Comparison of the Optimizers on the SQuAD Dataset for the RoBERTa model

The training and validation loss for all the opimizers can be found in Figure 4 and Figure 6. For the SQuAD Dataset, the AdamW and RMSProp optimizers converge the quickest. The SGD optimizer and the SGD with momentum optimizer get stuck in a local minima. They have a longer convergence time as compared to adaptive learning rate based optimizers like RMSProp and AdamW. The lowest loss was achieved by the AdamW optimizer.



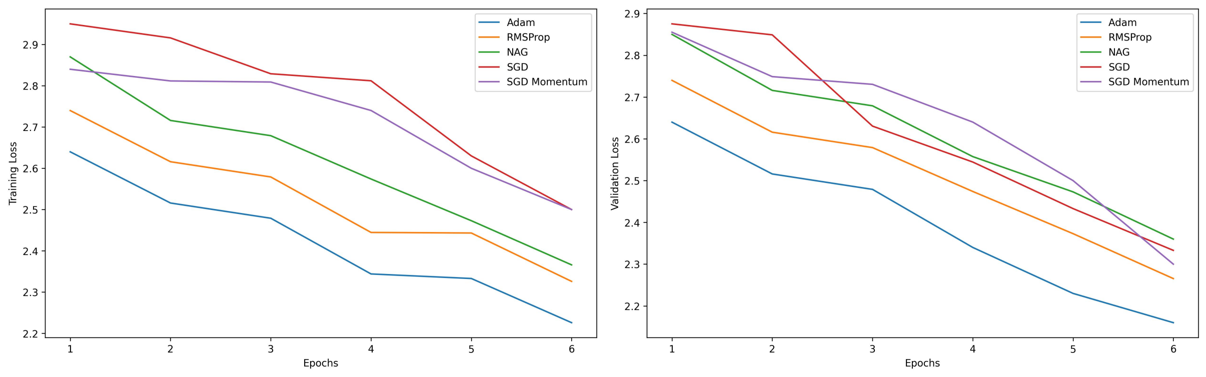
(a) Training Loss



(b) Validation Loss

Figure 5: Comparison of the Optimizers on the CoQA Dataset for the RoBERTa model

For the CoQA Dataset, the AdamW and RMSProp optimizers converge the quickest. The NAG optimizer gets stuck in a local minima and takes longer to converge. The SGD and SGD Momentum optimizer performs the worst and is not able to converge.



(a) Training Loss (b) Validation Loss

Figure 6: Comparison of the Optimizers on the FiQA Dataset for the LLama-3-8B model

For the FIQA Dataset, the AdamW and RMSProp optimizers converge the quickest. The NAG optimizer gets stuck in a local minima and takes longer to converge. The SGD and SGD Momentum optimizer performs the worst and is not able to converge.

6 Text Summarization

Text summarization is the process of breaking down lengthy text into digestible paragraphs or sentences. This method extracts vital information while also preserving the meaning of the text. This reduces the time required for grasping lengthy pieces such as articles without losing vital information. Text summarization is a natural language generation task that takes text as an input as well as generates text as the output.

The input sequence of tokens is mapped to a sequence of embeddings, which is then passed into the encoder. The encoder consists of a stack of blocks, each of which comprises two sub components: a self-attention layer followed by a small feed-forward network. Layer normalization is applied to the input of each subcomponent. After layer normalization, a residual skip connection adds each subcomponent's input to its output. A decoder transformer block is initialized. The decoder block is used for generating the summarized text. The self-attention mechanism in the decoder also uses a form of autoregressive or causal self-attention, which only allows the model to attend to past outputs. The output of the final decoder block is fed into a dense layer with a softmax output, whose weights are shared with the input embedding matrix. This is used to generate the final summarized text.

6.1 Datasets

The Multi-News and BillSum datasets were used for the Text Summarization task. The Text Summarization model generates the summarized text from the supplied text.

6.1.1 Multi-News

Multi-News (Fabbri et al. 2019) is the first large-scale multi-document summarization (MDS) news dataset, having 56,216 articles-summary pairs. It consists of news articles and human-written summaries of these articles from the site newser.com.

The dataset consists of:

- **Document**: Text of news articles.
- **Summary:** News summary.

6.1.2 BillSum

The BillSum dataset (Kornilova and Eidelman 2019) is the first corpus for automatic summarization of US Legislation containing 33,422 Bills. The corpus contains the text of bills and human-written summaries from the US Congress and California Legislature.

The dataset consists of:

- **Text**: Text present in the congressional or state bills.
- **Summary**: Summary of the Bills.

Text summarization refers to the technique of shortening long pieces of text. The intention is to create a coherent and fluent summary having only the main points outlined in the document.

6.2 Evaluation Method

The BART, DistilBART, and T5 models were trained for the text summarization task. The models were evaluated using the following metrics:

- **ROUGE-1**: It measures the match-rate of unigrams between the model output and a reference.
- **ROUGE-2**: It measures the match-rate of bigrams between the model output and a reference.
- **ROUGE-L**: Measures the number of matching 'l-grams' between the model-generated text and a reference.

6.3 Fine-tuning the models

For the Multi-News dataset, a baseline was set using the BART model. The BART model was fine-tuned using an AdamW optimizer with learning rate of 2e-5, and a batch-size of 32. The DistilBART model was also fine-tuned using an AdamW optimizer with learning rate of 2e-5 and a batch-size of 32. The model were trained for 6 epochs each. The performance of the model was evaluated on the basis of the ROUGE-1, ROUGE-2 and ROUGE-L scores on the test set. The DistilBART model achieved the best performance on the Multi-News dataset. Table 19 lists out the performance of the different BART architectures on the Multi-News dataset.

Model	ROUGE-1	ROUGE-2	ROUGE-L
DistillBART	32	10	19
BART	38	13	23

Table 19: Experiment results on the Multi-News Dataset

For the BillSum dataset, a baseline was set using the BART model. The BART model was fine-tuned using an AdamW optimizer with learning rate of 2e-5, and a batch-size of 8. The DistilBART and T5 models were also fine-tuned using an AdamW optimizer with learning rate of 2e-5 and a batch-size of 8. The models were trained for 5 epochs each. The performance of the model was evaluated on the basis of the ROUGE-1, ROUGE-2 and ROUGE-L scores on the test set. The DistilBART model achieved the best performance on the BillSum dataset. Table 20 lists out the performance of the different models on the BillSum dataset.

Model	ROUGE-1	ROUGE-2	ROUGE-L
DistillBART	28	9	21
T5	38	12	22
BART	40	17	25

Table 20: Experiment results on the BillSum Dataset

6.4 Optimizer Analysis

The optimizers were tuned for different parameters independently by a way of specific search spaces. Each parameter was first tuned on a large search space, and then a smaller and more refined search space was used to find the optimal value for training the model. We include both the initial search space used to refine the search. The final search space was used to generate the plots.

As shown in Table 2, the Learning Rate was tuned over an initial search space ranging from 1e-7 to 1e-3. The final search space was found to be 5e-6 to 1e-4. The optimal value of Learning Rate was found to be 5e-5 for Multi-News dataset and 3e-5 for BillSum dataset.

Optimizer	η (Learning Rate)	γ (Momentum)	α (Alpha)	β_1 (Beta1)	β_2 (Beta2)	ϵ (Epsilon)
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	3e-5	1e-1	1-1e-2	-	-	1e-5
AdamW	5e-5	1e-1	1-1e-2	1-1e-2	1-1e-2	1e-5

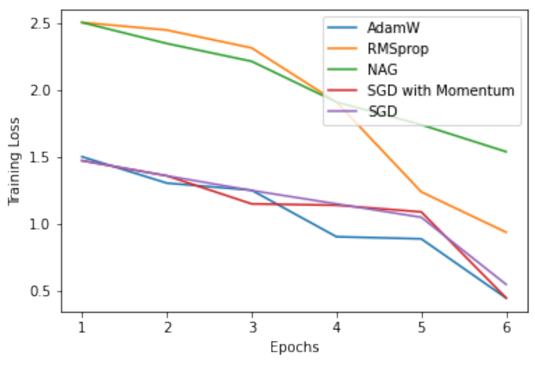
Table 21: Optimizer Hyperparameters for fine-tuning DistilBART on Multi-News Dataset.

As shown in Table 3, the Momentum was tuned over an initial search space ranging from 1e-4 to 1e-1. The final search space was found to be 1e-3 to 1. The optimal value of Momentum was found to be 1e-3 for both the datasets. As shown in Table 4, the Nesterov Momentum was tuned over an initial search space ranging from 1e-4 to 1.The final search space was found to be 1e-3 to 1e-1. The optimal value of Nesterov Momentum was found to be 1e-3 for both the datasets.

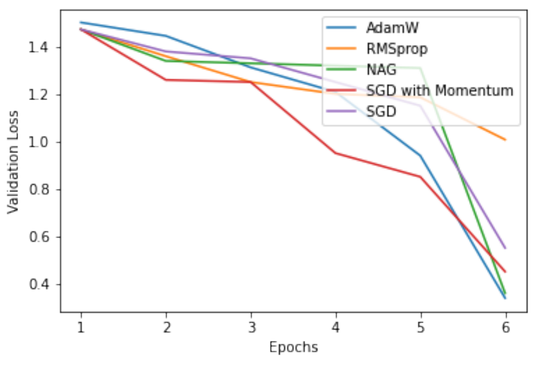
Optimizer	η (Learning Rate)	γ (Momentum)	α (Alpha)	β_1 (Beta1)	β_2 (Beta2)	ϵ (Epsilon)
SGD	0.01	-	-	-	-	-
SGD(Momentum)	1e-2	1e-3	-	-	-	-
NAG	1e-3	1e-3	-	-	-	-
RMSProp	2e-5	1e-1	1-1e-2	-	-	1e-5
AdamW	3e-5	1e-1	1-1e-2	1-1e-2	1-1e-2	1e-5

Table 22: Optimizer Hyperparameters for fine-tuning DistilBART on BillSum Dataset.

As shown in Table 5, the smoothing constant was tuned over an initial search space ranging from 0 to 1-1e-2. The final search space was found to be 0 to 1-1e-1. The optimal value of the smoothing constant was found to be 1-1e-2 for both the datasets. As shown in Table 6 the exponential decay rate of the first and second moments have been tuned over a initial search space ranging from 1-1e-1 to 1-1e-5 and 1-1e-1 to 1-1e-2. The optimal value of β_1 was found to be 1-1e-2 and the optimal value of β_2 (beta2) was found to be 1-1e-2 for both the datasets. The final values of the hyperparameters are shown in Table 21 and Table 22.



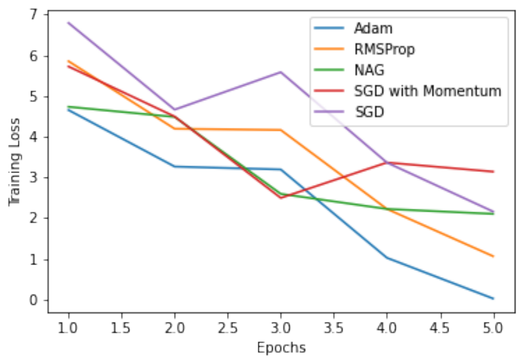
(a) Training Loss



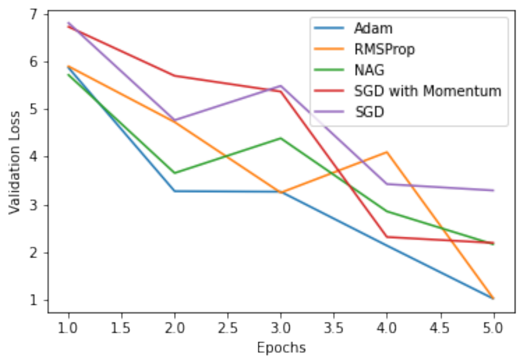
(b) Validation Loss

Figure 7: Comparison of the Optimizers on the Multi-News Dataset for the BART model

The training and validation loss for all the opimizers after tuning hyperparameters is shown in Figure 7 and Figure 8. For the Multi News Dataset, the optimizers AdamW and NAG converge the fastest. The RMSProp optimizer gets stuck in a local minima and takes longer to converge. The SGD and SGD with Momentum optimizers take longer to converge as compared to the adaptive learning rate based optimizers.



(a) Training Loss



(b) Validation Loss

Figure 8: Comparison of the Optimizers on the BillSum Dataset for the BART model

For the BillSum Dataset, the optimizers like AdamW and RMSProp converge the quickest. The NAG optimizer gets stuck in a local minima after three epochs and has a lower convergence time as compared to the other optimizers. The SGD and SGD with Momentum optimizers keep oscillating and are not able to converge.

7 Comparison of Hyperparameters

- For Sentiment Analysis on the Financial Phrase bank dataset, the FinBERT model was fine-tuned using an AdamW optimizer with a learning rate of $5e-5$, $\beta_1 = 0.90$, $\beta_2 = 0.90$, and $\epsilon = 10^{-5}$, using a batch size of 32 for 6 epochs.
- For Sentiment Analysis on the StockTwits dataset, the FinBERT model was fine-tuned using an AdamW optimizer with a learning rate of $3e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 32 for 6 epochs.
- For Sentiment Analysis on the FinGPT-Sentiment dataset, the Llama-3 model was fine-tuned using an AdamW optimizer with a learning rate of $3e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 16 for 6 epochs.
- For Question Answering on the SQuAD dataset, the RoBERTa model was fine-tuned using an AdamW optimizer with a learning rate of $5e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 32 for 6 epochs.
- For Question Answering on the CoQA dataset, the RoBERTa model was fine-tuned using an AdamW optimizer with a learning rate of $3e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 32 for 6 epochs.
- For Question Answering on the FiQA dataset, the Llama-3 model was fine-tuned using an AdamW optimizer with a learning rate of $3e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 16 for 6 epochs.
- For Text Summarization on the Multi-News dataset, the DistillBART model was fine-tuned using an AdamW optimizer with a learning rate of $2e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 32 for 6 epochs.
- For Text Summarization on the BillSum dataset, the DistillBART model was fine-tuned using an AdamW optimizer with a learning rate of $3e-5$, $\beta_1 = 0.99$, $\beta_2 = 0.99$, and $\epsilon = 10^{-5}$, using a batch size of 8 for 6 epochs.

8 GPT-3 Training

- To train the GPT-3 (Brown et al. 2020) model, the Adam optimizer was used with $\beta_1 = 0.9$ and $\beta_2 = 0.95$, and $\epsilon = 10^{-8}$. The global norm of the gradient was clipped at 1.0. A cosine decay for the learning rate down to 10% of its value, over 260 billion tokens (after 260 billion tokens, training continues at 10% of the original learning rate) was used. Then a linear learning rate warmup used over the first 375 million tokens.

- Also the batch size is gradually increased from a small value (32k tokens) to the full value over the first 4-12 billion tokens of training. Data was sampled without replacement during training (until an epoch boundary is reached) to minimize overfitting. All models use weight decay of 0.1 to provide a small amount of regularization.
- Mixed precision training is employed to reduce memory usage and speed up training, using both 16-bit and 32-bit floating-point types. Gradient accumulation is used to effectively increase the batch size by accumulating gradients over multiple smaller batches before performing weight updates.

9 Results from Optimizer Analysis

The difference between optimizers is entirely captured by the choice of update rule and hyperparameters. As a hyperparameter tuning protocol approaches optimality, a more expressive optimizer can never underperform any of its specializations. This can be shown by the inclusion relations between the optimizers as follows:

- SGD $\subseteq$ MOMENTUM $\subseteq$ RMSPROP
- SGD $\subseteq$ MOMENTUM $\subseteq$ ADAM
- SGD $\subseteq$ NESTEROV $\subseteq$ NADAM

When there exists an inclusion relationship between the optimizers , the more general optimizer can never be worse with respect to any metric of interest, provided the hyperparameters are sufficiently tuned to optimize that metric.

In our experiments, we chose the search space for each optimizer by running an initial set of experiments over a relatively large search space. In a typical case, we ran a single set of initial trials per optimizer to select the final search space. Given a search space, our tuning protocol sought to model a practitioner trying to achieve the best outcome with a fixed budget of trials. A feasible trial is any trial that achieves finite training loss.

Inclusion relationships between optimizers are very important in practice. More general optimizers never underperform their special cases. In particular, RMSProp and AdamW never underperformed SGD, Nesterov, or Momentum optimizers. We can conclude that the order of convergence of the optimizers is as follows:

$SGD \leq SGD(Momentum) \leq Adagrad \leq RMSProp \leq AdamW$

References

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman (2018). "GLUE: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461*. URL: `https://arxiv.org/abs/1804.07461`.

Alexander R Fabbri, Irene Li, Tianwei She, Suyi Li, and Dragomir R Radev (2019). "Multi-news: A large-scale multi-document summarization dataset and abstractive hierarchical model". In: *arXiv preprint arXiv:1906.01749*. URL: `https://arxiv.org/abs/1906.01749`.

Amil Merchant, Elahe Rahimtoroghi, Ellie Pavlick, and Ian Tenney (2020). "What happens to bert embeddings during fine-tuning?" In: *arXiv preprint arXiv:2004.14448*. URL: `https://arxiv.org/abs/2004.14448`.

Anastassia Kornilova and Vlad Eidelman (2019). "BillSum: A corpus for automatic summarization of US legislation". In: *arXiv preprint arXiv:1910.00523*. URL: `https://arxiv.org/abs/1910.00523`.

Andrea Galassi, Marco Lippi, and Paolo Torroni (2020). "Attention in natural language processing". In: *IEEE Transactions on Neural Networks and Learning Systems* 32.10, pp. 4291–4308. URL: `https://arxiv.org/abs/1902.02181`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in neural information processing systems* 30. URL: `https://arxiv.org/abs/1706.03762`.

Bernardino Romera-Paredes and Philip Torr (2015). "An embarrassingly simple approach to zero-shot learning". In: *International conference on machine learning*. PMLR, pp. 2152–2161. URL: `https://link.springer.com/chapter/10.1007/978-3-319-50077-5_2`.

Christopher D Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy (2020). “Emergent linguistic structure in artificial neural networks trained by self-supervision”. In: *Proceedings of the National Academy of Sciences* 117.48, pp. 30046–30054. URL: `https://www.semanticscholar.org/paper/Emergent-linguistic-structure-in-artificial-neural-Manning-Clark`.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. (2020). “Exploring the limits of transfer learning with a unified text-to-text transformer.” In: *J. Mach. Learn. Res.* 21.140, pp. 1–67. URL: `https://arxiv.org/abs/1910.10683`.

Dami Choi, Christopher J Shallue, Zachary Nado, Jaehoon Lee, Chris J Maddison, and George E Dahl (2019). “On empirical comparisons of optimizers for deep learning”. In: *arXiv preprint arXiv:1910.05446*. URL: `https://arxiv.org/abs/1910.05446`.

Dogu Araci (2019). “Finbert: Financial sentiment analysis with pre-trained language models”. In: *arXiv preprint arXiv:1908.10063*. URL: `https://github.com/ProsusAI/finBERT`.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. (2015). “Distilling the knowledge in a neural network”. In: *arXiv preprint arXiv:1503.02531* 2.7. URL: `https://arxiv.org/abs/1503.02531`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. (2023). “Llama 2: Open foundation and fine-tuned chat models”. In: *arXiv preprint arXiv:2307.09288*. URL: `https://arxiv.org/abs/2307.09288`.

Ian Tenney, Patrick Xia, Berlin Chen, Alex Wang, Adam Poliak, R Thomas McCoy, Najoung Kim, Benjamin Van Durme, Samuel R Bowman, Dipanjan Das, et al. (2019). “What do you learn from context? probing for sentence structure in contextualized word representations”. In: *arXiv preprint arXiv:1905.06316*. URL: `https://arxiv.org/abs/1905.06316`.

Iz Beltagy, Kyle Lo, and Arman Cohan (2019). “SciBERT: A pretrained language model for scientific text”. In: *arXiv preprint arXiv:1903.10676*. URL: `https://arxiv.org/abs/1903.10676`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805*. URL: `https://arxiv.org/abs/1810.04805`.

Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang (2020). “BioBERT: a pre-trained biomedical language representation model for biomedical text mining”. In: *Bioinformatics* 36.4, pp. 1234–1240. URL: `https://arxiv.org/abs/1901.08746`.

Kawin Ethayarajh (2019). “How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings”. In: *arXiv preprint arXiv:1909.00512*. URL: `https://arxiv.org/abs/1909.00512`.

Kawin Ethayarajh, David Duvenaud, and Graeme Hirst (2019). “Understanding undesirable word embedding associations”. In: *arXiv preprint arXiv:1908.06361*. URL: `https://arxiv.org/abs/1908.06361`.

Keita Kurita, Nidhi Vyas, Ayush Pareek, Alan W Black, and Yulia Tsvetkov (2019). “Measuring bias in contextualized word representations”. In: *arXiv preprint arXiv:1906.07337*. URL: `https://github.com/keitakurita/contextual_embedding_bias_measure`.

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. (2024). “Phi-3 technical report: A highly capable language model locally on your phone”. In: *arXiv preprint arXiv:2404.14219*. URL: `https://arxiv.org/abs/2404.14219`.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer (2019). “Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension”. In: *arXiv preprint arXiv:1910.13461*. URL: `https://arxiv.org/abs/1910.13461`.

Mukul Jaggi, Priyanka Mandal, Shreya Narang, Usman Naseem, and Matloob Khushi (2021). “Text mining of stocktwits data for predicting stock prices”. In: *Applied System Innovation* 4.1, p. 13. URL: `https://arxiv.org/abs/2103.16388`.

Natraj Raman, Pulkit Parikh, Lini Thomas, and Kamalakar Karlapalem (2022). “Semantics Extraction and Analytics from SEBI Regulations and Case Files: Industry-academia

Collaboration". In: *Workshop on Broadening Research Collaborations 2022*. URL: https://openreview.net/forum?id=CNsHRSPQ_3m.

Pekka Malo, Ankur Sinha, Pekka Korhonen, Jyrki Wallenius, and Pyry Takala (2014). "Good debt or bad debt: Detecting semantic orientations in economic texts". In: *Journal of the Association for Information Science and Technology* 65.4, pp. 782–796. URL: https://arxiv.org/abs/1307.5336.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang (2016). "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250*. URL: https://arxiv.org/abs/1606.05250.

Pranav Rajpurkar, Robin Jia, and Percy Liang (2018). "Know what you don't know: Unanswerable questions for SQuAD". In: *arXiv preprint arXiv:1806.03822*. URL: https://arxiv.org/abs/1806.03822.

Raj Sanjay Shah, Kunal Chawla, Dheeraj Eidnani, Agam Shah, Wendi Du, Sudheer Chava, Natraj Raman, Charese Smiley, Jiaao Chen, and Diyi Yang (2022). "WHEN FLUE MEETS FLANG: Benchmarks and Large Pre-trained Language Model for Financial Domain". In: *arXiv preprint arXiv:2211.00083*. URL: https://arxiv.org/abs/2211.00083.

Sander Blomme and Julie Dedeyne (2020). "Predicting the effect of 10-K, 10-Q and 8-K company reports on abnormal stock returns using FinBERT NLP methods." In: URL: https://lib.ugent.be/en/catalog/rug01:002837812.

Siva Reddy, Danqi Chen, and Christopher D Manning (2019). "Coqa: A conversational question answering challenge". In: *Transactions of the Association for Computational Linguistics* 7, pp. 249–266. URL: https://arxiv.org/abs/1808.07042.

Sneha Chaudhari, Varun Mithal, Gungor Polatkan, and Rohan Ramanath (2021). "An attentive survey of attention models". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 12.5, pp. 1–32. URL: https://arxiv.org/abs/1904.02874.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. (2019). "Huggingface's transformers: State-of-the-art natural language processing". In: *arXiv preprint arXiv:1910.03771*. URL: https://arxiv.org/abs/1910.03771.

Tobias Daudert (2021). "Exploiting textual and relationship information for fine-grained financial sentiment analysis". In: *Knowledge-Based Systems* 230, p. 107389. URL: https://www.sciencedirect.com/science/article/pii/S0950705121006511.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. (2020). "Language models are few-shot learners". In: *Advances in neural information processing systems* 33, pp. 1877–1901. URL: https://arxiv.org/abs/2005.14165.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108*. URL: https://arxiv.org/abs/1910.01108.

Vivek Srivastava, Stephen Pilli, Savita Bhat, Niranjan Pedanekar, and Shirish Karande (2021). "What BERTs and GPTs know about your brand? Probing contextual language models for affect associations". In: *Proceedings of Deep Learning Inside Out (DeeLIO): The 2nd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pp. 119–128. URL: https://aclanthology.org/2021.deelio-1.12/.

Yi Yang, Mark Christopher Siy Uy, and Allen Huang (2020). "Finbert: A pretrained language model for financial communications". In: *arXiv preprint arXiv:2006.08097*. URL: https://arxiv.org/abs/2006.08097.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov (2019). "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692*. URL: https://arxiv.org/abs/1907.11692.

Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata (2018). "Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly". In: *IEEE transactions on pattern analysis and machine intelligence* 41.9, pp. 2251–2265. URL: https://arxiv.org/abs/1707.00600.

Zhiyu Chen, Shiyang Li, Charese Smiley, Zhiqiang Ma, Sameena Shah, and William Yang Wang (2022). "Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering". In: *arXiv preprint arXiv:2210.03849*. URL: `https://arxiv.org/abs/2210.03849`.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. (2021). "Finqa: A dataset of numerical reasoning over financial data". In: *arXiv preprint arXiv:2109.00122*. URL: `https://arxiv.org/abs/2109.00122`.