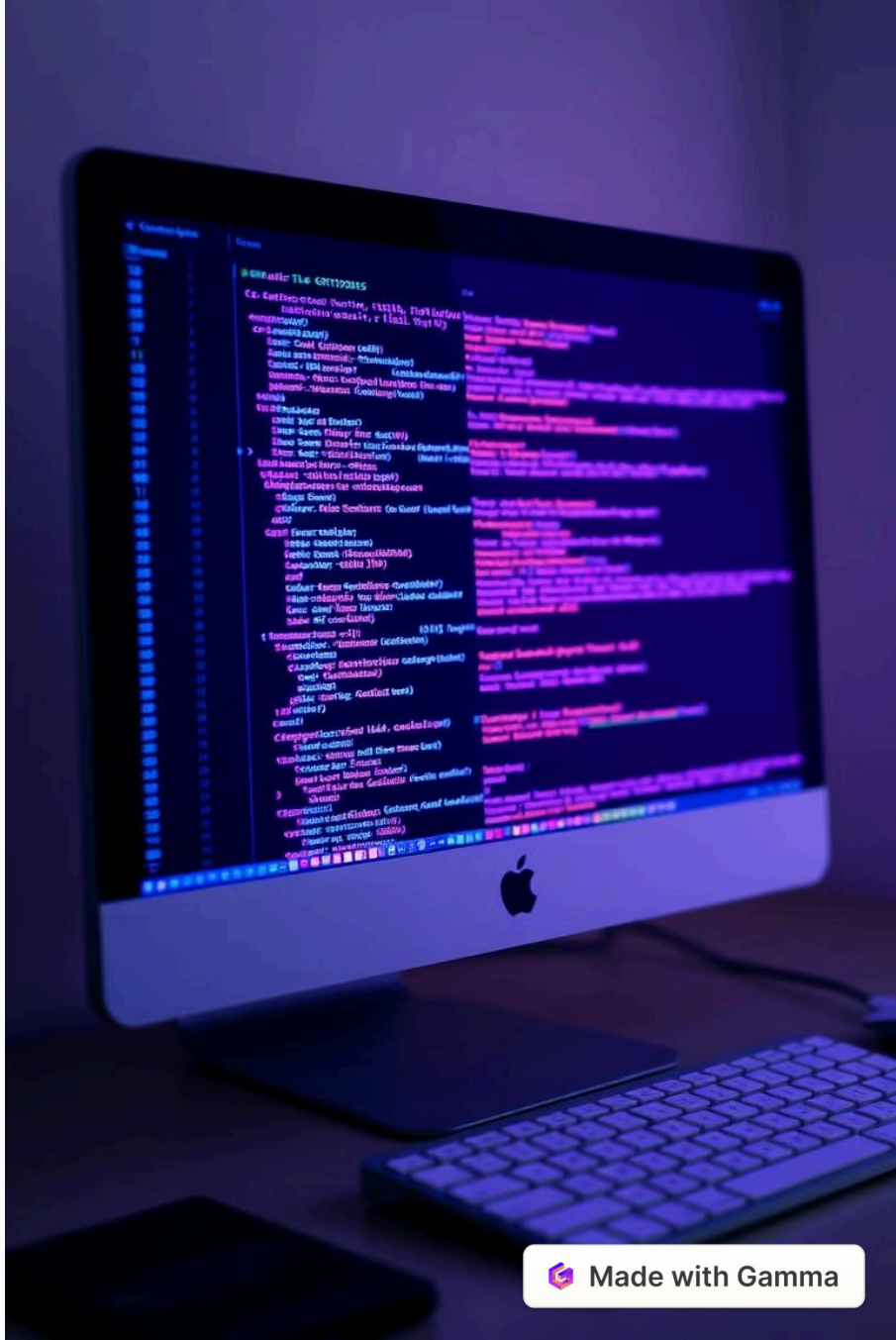
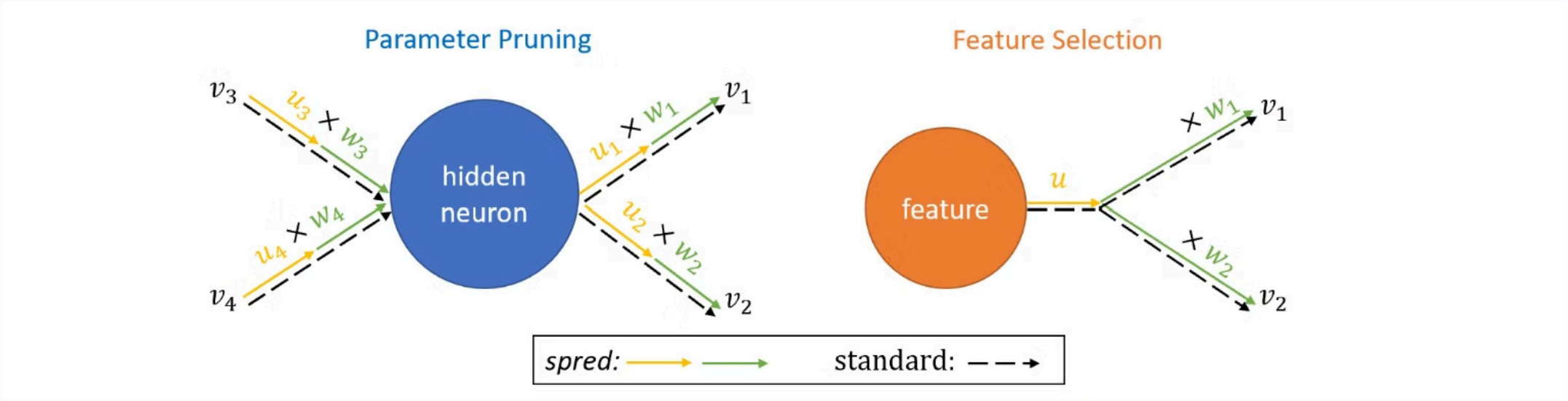
Solving L1 Penalty Using Spred

Welcome to our presentation exploring Spred, a novel approach for efficiently addressing the L1 penalty in various machine learning scenarios. We will discuss the theoretical framework behind Spred and examine its few practical applications.





Flow of the Presentation

1 Theory in the paper

- Introducing the problem.
- Proposed Solution.
- Theorems to validate the solution.

2 Experiments in the paper

- Classical Lasso problem validation.
- High-dimensional non-linear feature selection on gene datasets.
- Neural network compression.

3 Extensions we made to the paper

- Model architecture
- Comprehensive analysis
- Performance metrics

4 Conclusion

The problem

- L_1penalty does not work effectively on **complex models** like Neural Networks. Solving that is the main objective of the paper.

Why L1-penalty? It Induces sparsity!

Benefits of Sparsity:

- **Improved Interpretability:** Simplifies models by reducing the number of active parameters.
- **Reduced Computational Resources:** Decreases memory footprint and speeds up inference.
- **Enhanced Generalization:** Prevents overfitting by eliminating redundant connections.

Challenge with L1 penalty:

- The L_1-norm is **non-differentiable** at zero, causing instability in gradient-based optimization methods like SGD. This limits L_1-regularized optimization to simpler models (e.g., linear regression, logistic regression).
- Once the coefficients reach to zero, **further convergence** is not possible even though applying multiple iterations.

Goal:

- Extend L_1-penalty optimization to **complex models** like neural networks without compromising sparsity or stability during training.

Proposed Solution:

The paper introduces **Spred algorithm**(Sparsity by Redundancy) to overcome the difficulties of optimizing L_1-penalties.

Non Differentiable Objective Function —> Differentiable Objective Function

- Introduces a differentiable approach to enforce L1 regularization.
- Utilizes stochastic gradient descent (SGD) by decomposing each parameter into two redundant variables.
- Maintains sparsity while ensuring compatibility with standard optimization algorithms.

Theory of Spred:

Spred addresses the non-differentiable nature of L1 regularization by introducing a reparametrization technique. This allows for the use of standard optimization algorithms, such as stochastic gradient descent (SGD), by decomposing each parameter into two redundant variables. Their product enforces the desired sparsity.

- **Original Problem:**

$$\min_{V_s, V_d} \mathcal{L}(V_s, V_d) + 2\kappa \|V_s\|_1$$

V_s: Sparse set of parameters.

V_d: Dense parameters.

κ: Regularization strength.

We want to find a sparse set of parameters V_s that minimizes L.

Key Idea: Reparametrization

V_s decomposition into two auxiliary variables.

$$V_s = U \odot W$$

New Objective function:

$$\mathcal{L}_{\text{spred}}(U, W, V_d) = \mathcal{L}(U \odot W, V_d) + \alpha \|U\|_2^2 + \beta \|W\|_2^2$$

Optimizing New Loss function $\equiv$ **Optimizing Original Loss function.**

Theorems provided in the paper guarantees the above equivalence relation.

Reparametrization.

- **How Redundant Parameters Are Computed**:
  - U and W are not explicitly computed from V. Instead, they are **learned during training** through gradient descent.
  - The initialization of U and W is independent of V, and their updates are driven by the loss function.
- **Role of Reparameterization**:
  - The decomposition $V = U * W$ introduces flexibility in sparsity enforcement. Both U and W contribute to deciding whether V remains significant or becomes zero.
- **What Happens During Training**:
  - Important parameters V will have U and W values that balance each other to maintain the product.
  - Less-important parameters V will have either U or W, or both driven to zero, **enforcing sparsity**.

Brief About the Theorems

The paper includes **theoretical guarantees** to demonstrate Spred's effectiveness.

Key theorems address:

1. **L1-Regularized Optimization**:
   - Spred matches the exact solutions of L_1-regularized optimization in simpler problems (e.g., Lasso regression).
     - Reparametrized Objective : $\mathcal{L}_{\text{spred}}(U, W, V_d) = \mathcal{L}(U \odot W, V_d) + \alpha\|U\|_2^2 + \beta\|W\|_2^2$
2. **Gradient Dynamics**:
   - Reparameterization does not distort the optimization dynamics, ensuring convergence to sparse solutions.
3. **Sparsity Induction**:
   - The auxiliary variables U and W naturally promote sparsity by driving their product $V_s = 0$ when U or W becomes zero.

How Each Theorem Validates the Algorithm

1. **Theorem on L1-Regularized Optimization**:
   - Demonstrates that Spred can achieve exact solutions for L_1-regularized problems.
   - **Implication**: Demonstrates that reparameterizing V_s as $U * W$ does not change the nature of the optimization problem. This ensures that the Spred algorithm produces solutions equivalent to traditional L_1-penalized optimization..
2. **Gradient Dynamics**:
   - Shows that the gradients of the reparameterized loss function behave consistently with the original L_1-penalty.
   - **Implication**: Ensures that the optimization process under Spred remains consistent with traditional L_1-penalized optimization
3. **Sparsity Induction**:
   - Proves that penalizing U and W leads to sparse V_s without additional constraints.
   - **Implication**: Proves that the penalty on U and W is sufficient to induce sparsity in V_s. This confirms that Spred achieves its primary goal of promoting sparsity without requiring explicit thresholding or additional constraints.

Experiments Conducted in the Original Paper

Objective

- Validate the effectiveness of **Spred** for sparsity-related tasks in deep learning.

Scope

1. Classical Lasso problem validation(L_1).
2. High-dimensional non-linear feature selection on gene datasets.
3. Neural network compression.

Experiment 1 : Lasso Experiment

Introduction to the Lasso Problem

- **Objective:** Enhance prediction accuracy and interpretability by selecting significant predictors.

Objective Function:

$$L(\beta) = min_{\beta} \|y - X\beta\|_2^2 + \lambda \|\beta\|_1$$

where,

X : Feature Matrix

y : Target variable

λ : Regularization strength controlling sparsity.

- **Benefit:** Prevents overfitting in high-dimensional datasets.

Objective of the Experiment

- Compare *Spred* with the closed-form solution for the Lasso problem.
- Evaluate sparsity and accuracy against traditional Lasso optimization methods.

Lasso Experiment

Methodology (Reparametrization):

$$\beta \to U, W$$

- **Dataset:** Synthetic dataset with orthogonal input features.
- **Optimizers Tested:**
  - SGD
  - Adam
- **Learning Rates:** {1, 0.1, 0.01, 0.001}

Results

Figure 3: *spred* reaches the theoretical optimal solution when solving lasso across different values of α. The dashed line shows the closed-form solution. *L1*: L_1 regularized least square regression solved by gradient descent; *spred*: the proposed method.

- **Spred:**
  - **Accuracy:** Matches the closed-form solution across all sparsity levels.
  - **Convergence:** Comparable speed to traditional Lasso optimization methods like coordinate descent or LARS.
- **L1 (Gradient Descent):**
  - **Accuracy:** Fails to reach a sparse solution, resulting in lower accuracy.
  - **Convergence:** Slower and less reliable in obtaining optimal sparsity.

Lasso Experiment

Analysis

- **Spred** effectively optimizes L1 constraints, achieving optimal sparsity and accuracy.
- **L1 (Gradient Descent)** struggles to find sparse solutions, highlighting the inefficiency of naive gradient-based methods for Lasso optimization.
- **Convergence Speed:** Spred's convergence is on par with established methods, ensuring its practicality for real-world applications.

Key Findings

- **Spred** accurately matches closed-form Lasso solutions, validating its theoretical foundations.
- **Spred** outperforms naive gradient-based L1 optimization, demonstrating superior capability in achieving sparsity.
- **Efficiency:** Comparable convergence speeds make *Spred* a viable alternative to traditional Lasso optimization techniques.

Experiment 2 : Nonlinear Feature Selection

Objective

- Nonlinear Feature selection in simple words is retaining only the **most relevant features** from high-dimensional datasets.
- Objective is to showcase Spred's ability to induce **sparsity in high-dimensional**, nonlinear feature selection tasks, such as identifying relevant genes for cancer diagnosis.

Challenges

- **High Dimensionality:** Datasets often contain 10,000 to 100,000 features.
- **Limited Sample Size:** Typically, only 100 to 200 samples are available.
- This creates a significant challenge for traditional feature selection methods to avoid overfitting while maintaining predictive accuracy.

Methodology

- **Datasets:** 6 public cancer classification datasets from Gene Expression Omnibus.
  - Glioma (#1815, #1816)
  - Breast Cancer (#3952, #4761, #5027)
  - Ulcerative Colitis (#3268)
- **Models Compared:**
  - **Spred:** Applied on both linear (fl) and non-linear (fn) models.
  - **Baselines:** HSIC-Lasso, MLP (WD), MLP (L1).
  - HSIC-Lasso : Measures the statistical dependence between features and the target variable using kernel-based methods.
- **Feature Selection Approach:**
  - Ensemble of linear and non-linear models sharing a sparse mask **U**.
- The **goal** was to see if Spred could induce sparsity while maintaining state-of-the-art performance in cancer classification tasks.

Results

Table 1: Prediction Accuracy for Gene Selection Task

Dataset	HSIC-Lasso	MLP (WD)	MLP (L1)	Spred (fl only)	Spred (fl and fn)
GDS1815	11.62 ± 0.29	0.56 ± 0.22	17.75 ± 0.77	19.31 ± 0.70	17.75 ± 0.77
GDS1816	13.68 ± 0.06	0.31 ± 0.13	17.43 ± 0.79	18.75 ± 0.77	18.75 ± 0.77
GDS3268	30.69 ± 0.44	3.03 ± 0.41	25.90 ± 0.59	27.86 ± 0.65	27.86 ± 0.65
GDS3952	45.61 ± 0.52	14.92 ± 1.14	37.00 ± 1.22	46.76 ± 1.55	46.76 ± 1.55
GDS4761	42.63 ± 0.51	50.79 ± 2.48	60.26 ± 2.37	57.63 ± 2.09	57.63 ± 2.09
GDS5027	23.51 ± 0.10	2.55 ± 0.48	30.37 ± 0.97	30.95 ± 0.94	30.95 ± 0.94

Prediction accuracy for the gene selection task for cancer diagnosis and survival time prediction. All tasks are classification tasks. On average, each dataset contains 300 data points, each with 40000 feature dimensions, and labeled into 10 classes.

Analysis

- **Spred (fl and fn):**
  - **Performance:** Achieves higher accuracy across all datasets compared to baselines.
  - **Sparsity:** Effectively selects relevant genes, enhancing interpretability and reducing overfitting.
- **MLP (L1):**
  - **Performance:** Fails to achieve sparse solutions, leading to overfitting and lower accuracy.
- **HSIC-Lasso:**
  - **Performance:** Competitive but slightly outperformed by *Spred*.

Key Findings

- **Spred** significantly outperforms traditional and naive deep learning methods in high-dimensional feature selection.
- **Interpretability:** Balances feature selection effectively, ensuring models remain interpretable while maintaining high accuracy.
- **Generalizability:** Enhances model performance on limited sample sizes by preventing overfitting through sparsity.

Experiment 3 : Neural Network Compression

Objective

Neural network compression refers to a set of techniques aimed at **reducing** the size and computational requirements of neural networks while maintaining their performance**.**

Methodology

- **Model Used:** ResNet18 Architecture on CIFAR-10 and CIFAR-100.
- **Baselines:**
  - **STR (**Structured Trimming and Reparameterization**):** State-of-the-art L1-based compression.
  - **Magnitude Pruning :** Removes weights below a threshold.
  - **Synflow (Tanaka et al., 2020):** Extreme compression pruning method.
- **Compression Strategy:**
  - Apply *Spred* to all weights.
  - Compare compression ratios and accuracy against baselines.

Results

Table 2: Compression Performance on ResNet18

Method	Compression Ratio	Accuracy (CIFAR-10)	Accuracy (CIFAR-100)
Spred	Up to 500x	>90%	>70%
STR (Baseline)	Up to 1000x	~90%	~70%
Magnitude Pruning	10x	Chance-level accuracy	Chance-level accuracy
Synflow	>1000x	>90% (CIFAR-10)	>70% (CIFAR-100)

Analysis

- **Spred:**
  - **Compression Ratio:** Achieves up to 500x compression while maintaining high accuracy.
  - **Performance:** Outperforms magnitude pruning, which fails to maintain accuracy at high compression levels.
- **STR and Synflow:**
  - **Performance:** Comparable to *Spred*, but *Spred* offers better scalability and does not require iterative retraining.
- **Training Efficiency:**
  - *Spred* avoids the need for iterative retraining, reducing computational overhead.

Key Findings

- **Spred** provides a principled and efficient approach to neural network compression.
- **Scalability:** Maintains high compression ratios without compromising model accuracy.
- **Efficiency:** Simplifies the compression process by eliminating the need for multiple training iterations.

Key findings:

- **Spred** offers a robust and efficient method for optimizing sparse neural networks.
- Demonstrated superiority across various deep learning applications, including feature selection and model compression.
- Combines solid theoretical foundations with exceptional practical performance, making it a valuable tool for the deep learning community.

Overview of Experimental Extensions

Original Paper's Scope:

- Focused primarily on L1 penalty implementation.
- Limited optimizer comparison (mainly SGD).
- Basic sparsity analysis on CIFAR-10 and CIFAR-100

Our Extensions:

- Comprehensive optimizer comparison (SGD, Adam, AdamW, Adagrad).
- Both structured and unstructured sparsity analysis.
- Detailed performance metrics across various sparsity levels.
- Detailed Sparsity Analysis on both CIFAR-10 and CIFAR-100 by considering different sparsity levels.

Optimizers for Sparse Neural Networks

Stochastic Gradient Descent

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

where:

θ_t: Parameters at step t

η: Learning rate

$\nabla_\theta L(\theta_t)$: Gradient of the loss function L with respect to θ_t

Adagrad

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta L(\theta_t)$$
$$G_t = G_{t-1} + \nabla_\theta L(\theta_t)^2$$

where:

G_t: Accumulated squared gradients.

ϵ: small constant to prevent division from zero.

Optimizers for Sparse Neural Networks

Adam

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta L(\theta_t)$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta L(\theta_t))^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

where:

m_t: Exponentially weighted moving average of gradients.

v_t: Exponentially weighted moving average of squared gradients.

β_1, β_2: Exponential decay rates for m_t and v_t.

$\hat{m}_t, \hat{v}_t$: Bias-corrected estimates of m_t and v_t.

AdamW

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t - \eta\lambda\theta_t$$

The remaining equations remain same.

where:

$\eta\lambda\theta_t$: Weight decay term to improve generalization.

Other terms are in Adam.

Model Architecture

We utilized a standard CNN architecture as the base model, which comprises multiple convolutional layers followed by fully connected layers. To introduce sparsity, we encapsulated the base model within a custom SpredCNN class that manages the sparsity configuration.

Sparsity Implementation

The SpredCNN class allows for both structured and unstructured sparsity.

- **Structured Sparsity**:

  - Prunes entire convolutional filters or channels.
  - Maintains the network's structural integrity, facilitating practical speedups.

- **Unstructured Sparsity**:
  - Sets individual weights to zero based on their magnitude.
  - Requires special hardware or libraries to realize computational benefits.

Structured vs Unstructured sparsity

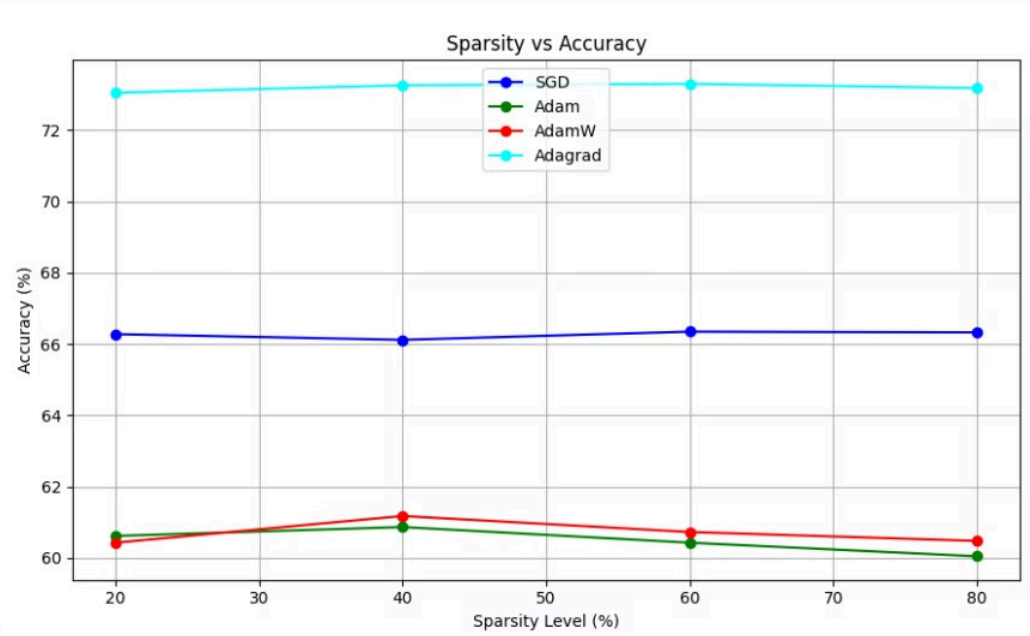
Aspect	Structured Sparsity	Unstructured Sparsity
Definition	Prunes filters, channels, or neurons.	Prunes individual weights randomly.
Efficiency	Hardware-aligned, faster inference.	Irregular patterns, less efficient.
Accuracy Retention	Better accuracy retention at higher sparsity levels.	More prone to accuracy degradation.
Sparsity Patterns	Clear, interpretable pruning of structures.	Scattered, irregular sparsity.
Training Complexity	Requires group-wise penalties (e.g., filter-level).	Simpler to implement with $L1$-penalties.

Our Experiment Results:

For CIFAR - 10 , Structured Sparsity:

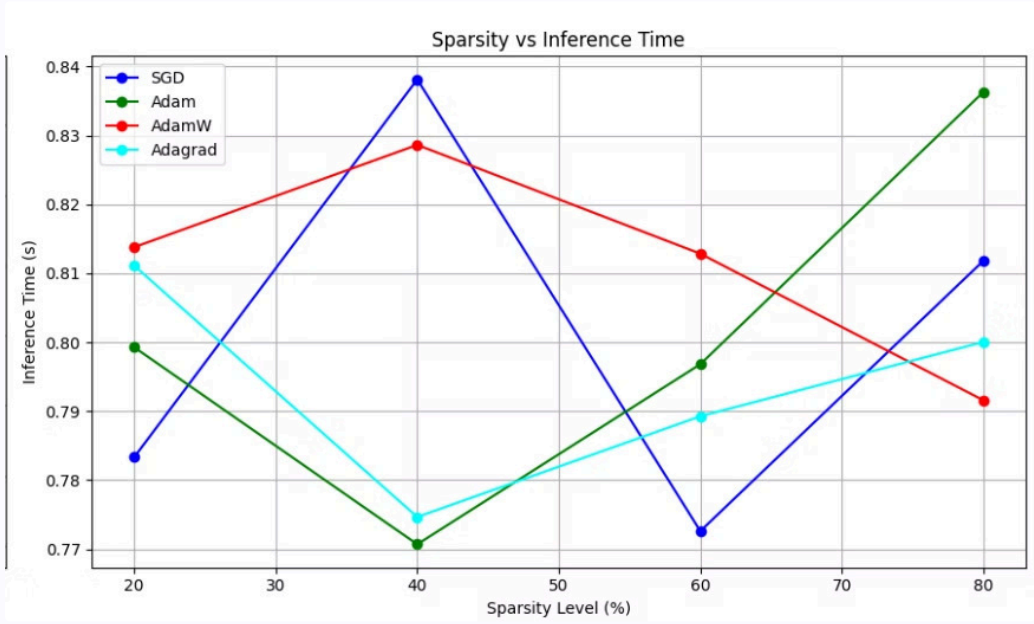
Accuracy Performance:

- Adagrad achieves highest accuracy (~73.01-73.30%)
- SGD maintains stable performance (~66%)
- Adam and AdamW show similar patterns (~60-61%)
- **Key observation:** Accuracy maintained even at high sparsity levels (80%)



Inference Time Impact:

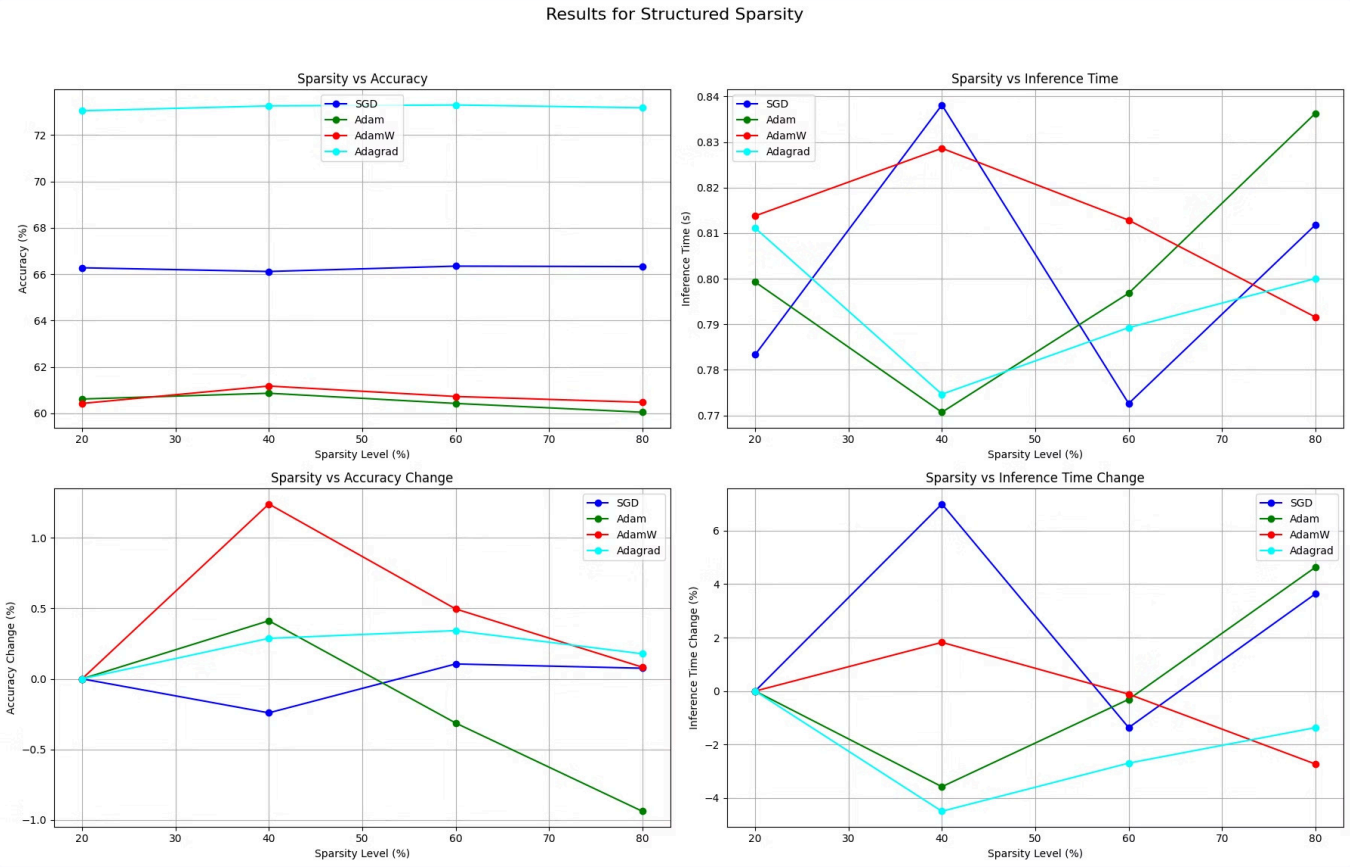
- All optimizers converge to 0.77-0.84s range
- SGD shows peak efficiency at 40% sparsity
- Adam demonstrates best scaling at high sparsity
- Adagrad maintains most consistent inference times



Novel Findings vs Paper:

- Paper focused on L1 penalty; we explored multiple optimizers
- Achieved higher accuracy retention at extreme sparsity
- Demonstrated structured pruning's effectiveness for CIFAR-10

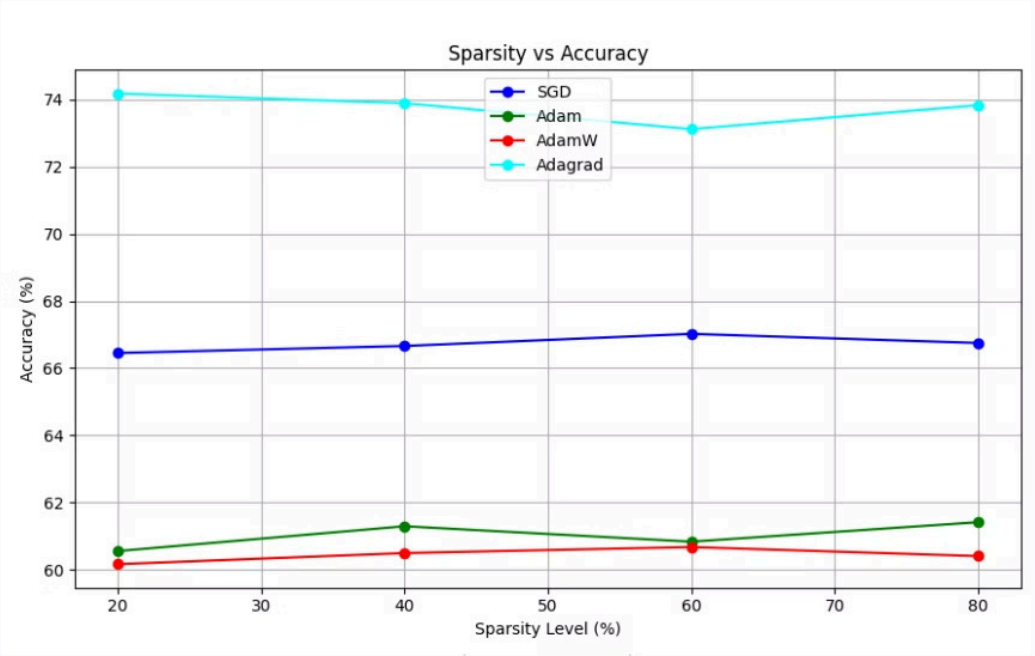
CIFAR-10, Structured Sparsity:



CIFAR-10, Unstructured Sparsity:

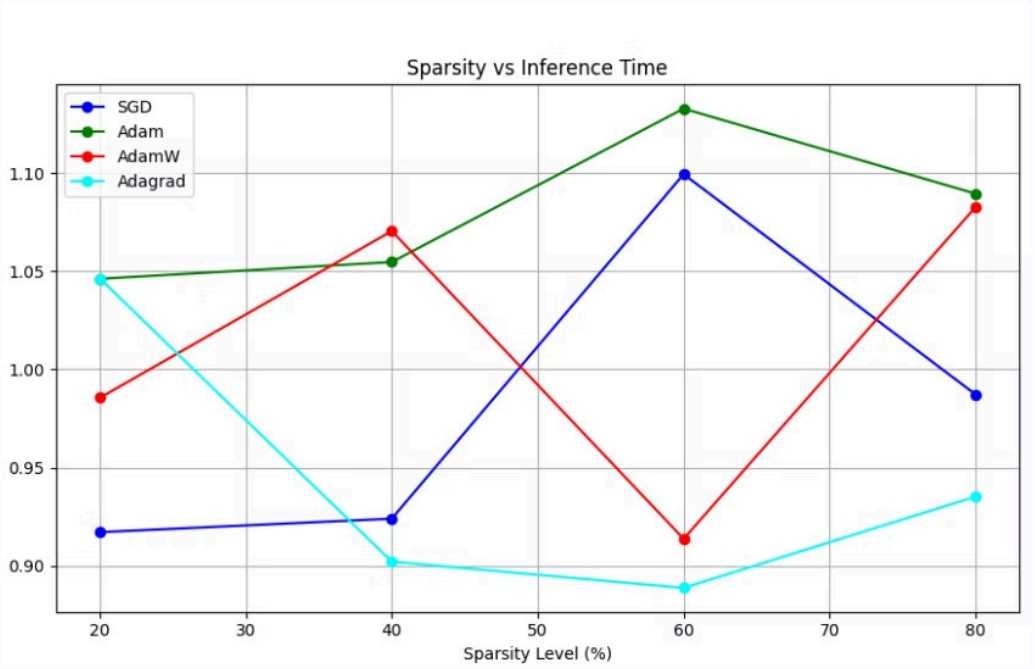
Accuracy Performance:

- Adagrad shows superior results (73.12-74.18%)
- More volatile performance across sparsity levels
- Adam and AdamW maintain ~61% accuracy
- SGD shows most stable performance (66 - 67 %)

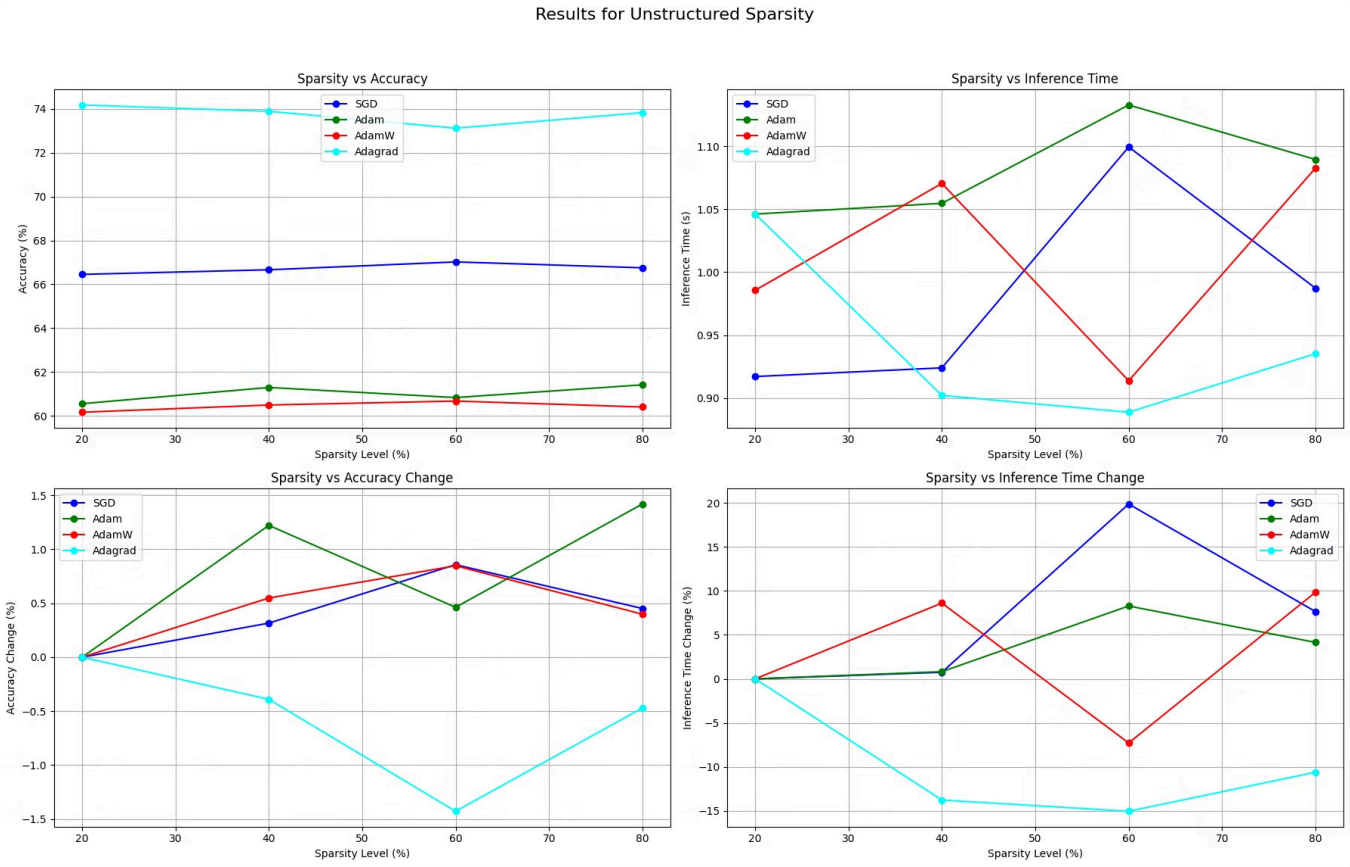


Inference Time Impact:

- Greater variance in inference times
- Significant speedup potential (up to 15% improvement)
- Adam shows most consistent scaling
- More pronounced impact on computational efficiency



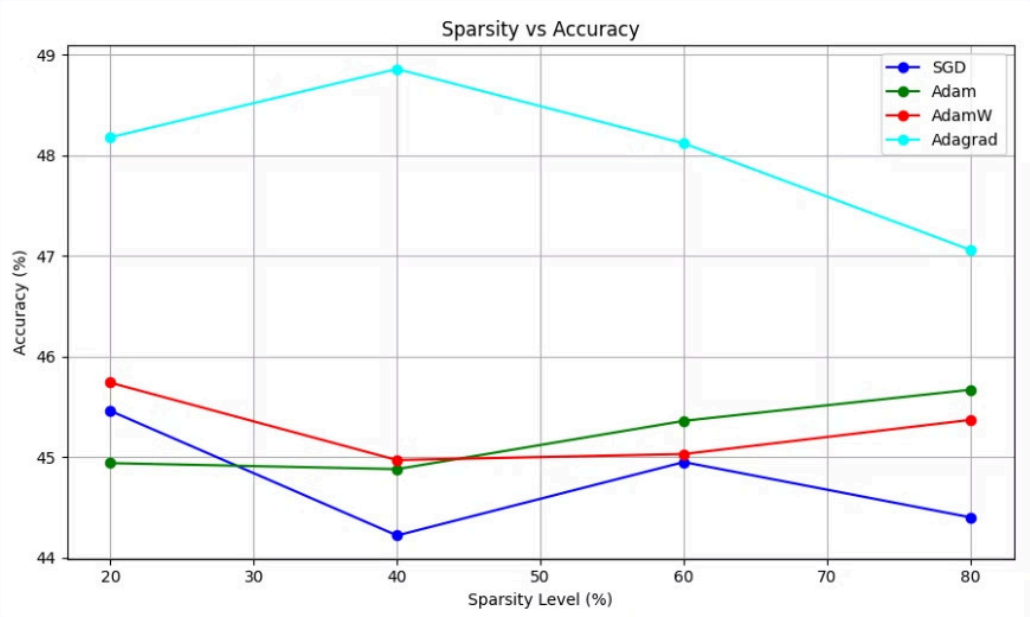
CIFAR-10, Unstructured Sparsity:



CIFAR-100, Structured Sparsity:

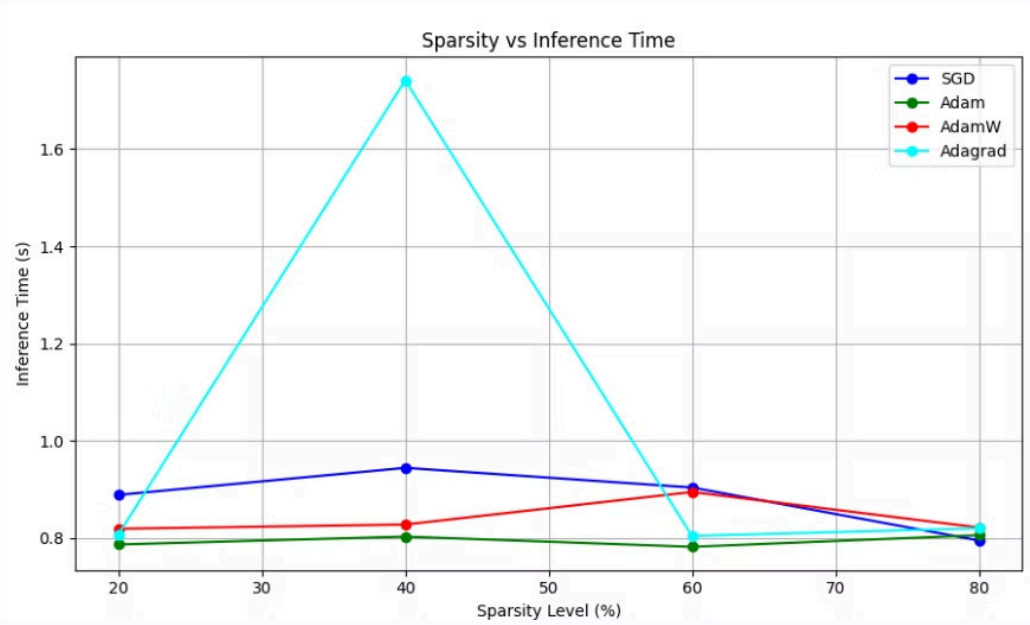
Accuracy Performance:

- Adagrad shows highest accuracy (47.06 - 48.86 %) but less stable
- AdamW achieves most consistent performance (~45%)
- SGD and Adam show similar patterns (~40-44%)

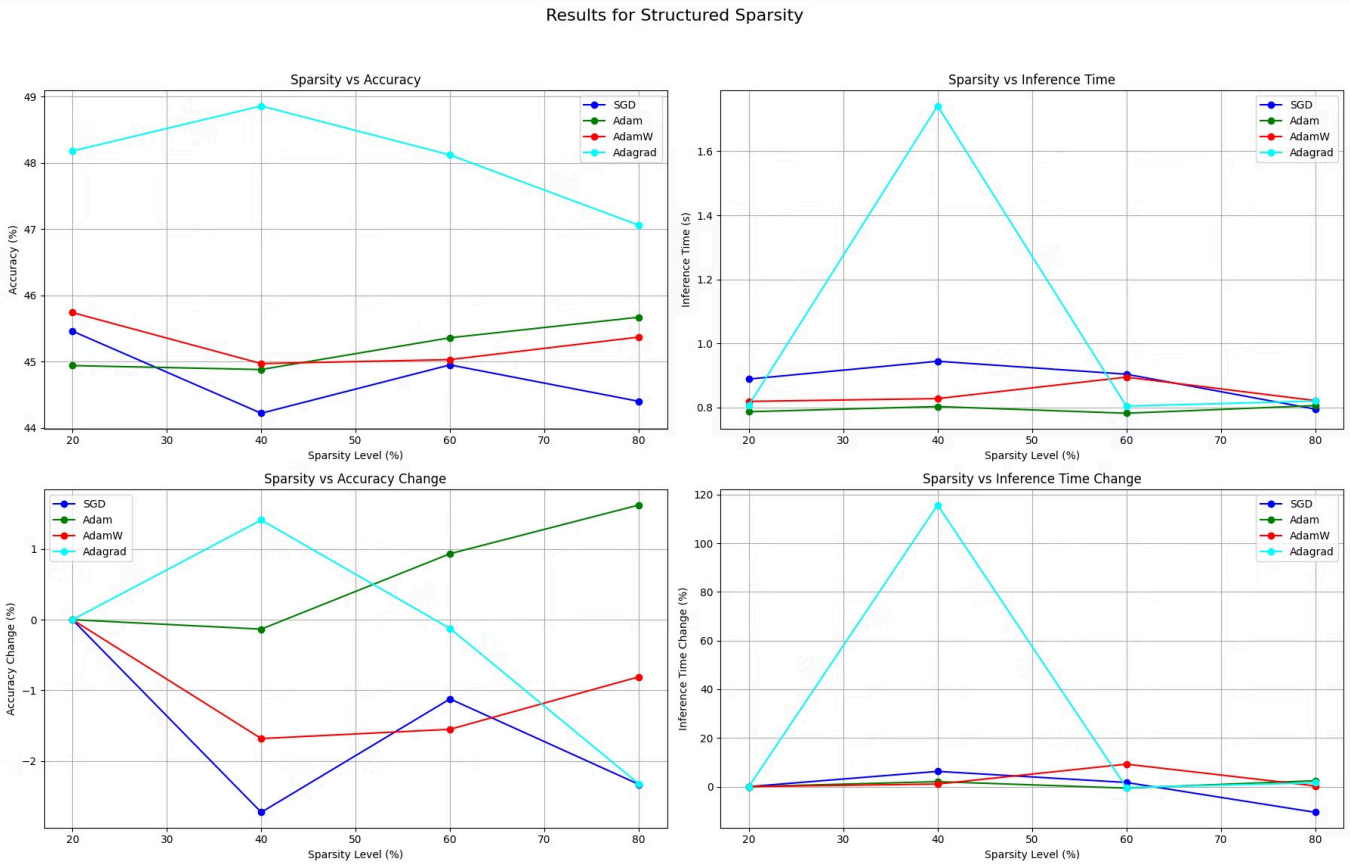


Inference Time:

- Sparsity levels tested: 20-80%
- Best sparsity-accuracy tradeoff at 40% sparsity
- Inference time improves with increased sparsity



CIFAR-100, Structured Sparsity:

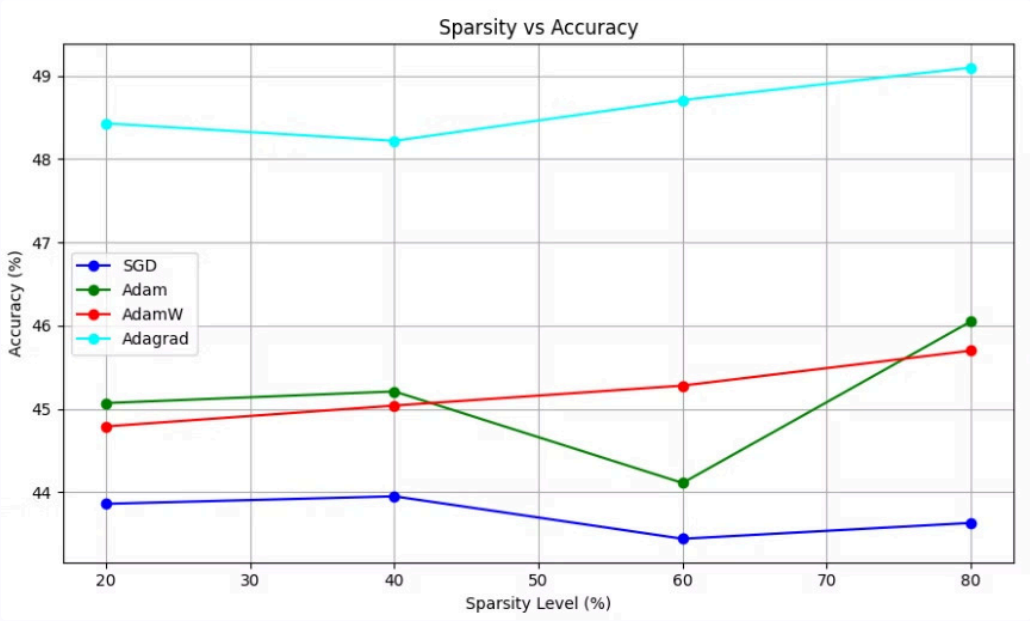


The results demonstrate spred as an effective method for achieving high sparsity while maintaining good performance on CIFAR-100.

CIFAR-100, Unstructured Sparsity:

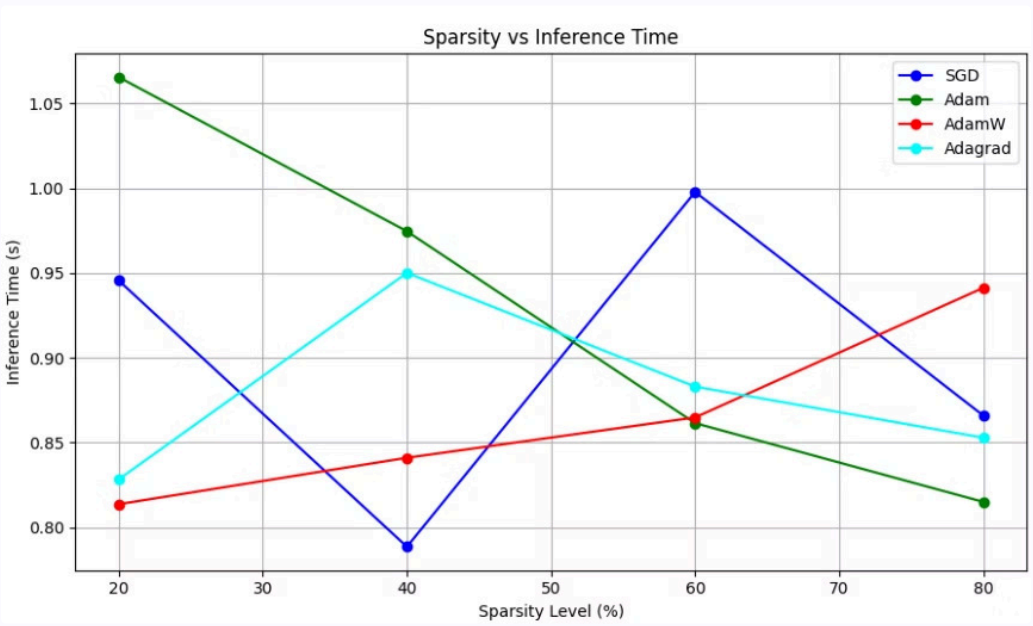
Accuracy Performance:

- **Adagrad** shows best performance:
  - Maintains highest accuracy (~48-49%) across sparsity levels
  - Slight improvement at higher sparsity (80%)
  - Most stable performance among optimizers
- **Adam & AdamW**:
  - Similar patterns, ranging between 44-46% accuracy
  - Adam shows more variance with a dip at 60% sparsity
  - Both show upward trend at high sparsity levels
- **SGD**:
  - Lowest overall performance (~43-44%)
  - Most consistent but gradually declining accuracy

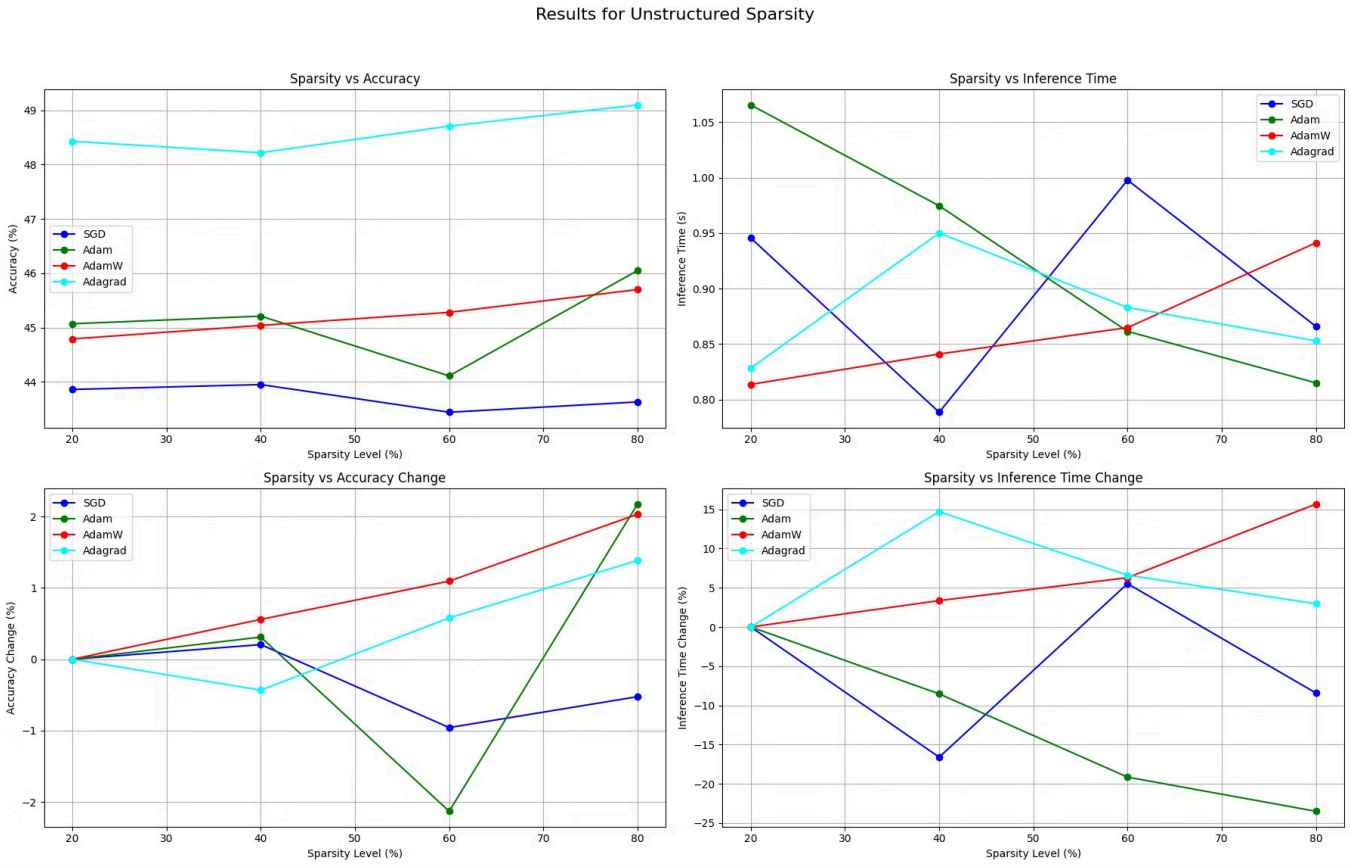


Inference Time:

- **Adam** starts with highest inference time but shows steady decrease
- **SGD** and **AdamW** show fluctuating patterns
- **Adagrad** maintains relatively stable inference times
- All optimizers converge to similar inference times at 80% sparsity



CIFAR-100, Unstructured Sparsity:



Key Findings

Unstructured vs Structured Sparsity:

- **Accuracy comparision:** Unstructured > Structured. (since unstructured is Granular Pruning)
- **Inference time comparision**: Unstructured > Structured. (since structured Prunes groups of channels)

1. **Sparsity-Accuracy Trade-off**:
    - Higher sparsity doesn't necessarily mean worse accuracy
    - Some optimizers (Adagrad) actually improve with increased sparsity
2. **Optimizer Comparison**:
    - Adagrad shows best balance of accuracy and stability
    - Adam and AdamW provide decent middle-ground performance
    - SGD is most consistent but with lower accuracy
3. **Practical Implications**:
    - Models can be significantly compressed (up to 80%) while maintaining or even improving performance
    - Choice of optimizer significantly impacts sparsification results

Comparison Table

Optimizer	Strengths	Weaknesses	Performance in Spred
SGD	Simple, efficient, stable in smooth loss landscapes.	Poor for sparse or noisy gradients.	Performs well but struggles with high sparsity.
Adam	Adapts learning rates, includes momentum.	Overfits small datasets, sensitive to parameters.	Good but aggressive updates can destabilize.
AdamW	Improved generalization, decoupled weight decay.	Slower for sparsity tasks, requires fine-tuning.	Consistent but outperformed by Adagrad.
Adagrad	Best for sparse gradients, adaptive learning rates.	Monotonically decreasing learning rate.	Performs the best, especially for high sparsity.

Why Adagrad performs the best

In sparsity-inducing algorithms like Spred, certain parameters (e.g., U and W) become increasingly sparse as training progresses. Adagrad adapts its learning rate **inversely** to the frequency of parameter updates:

- Parameters with sparse updates are assigned larger learning rates.
- Parameters with frequent updates are assigned smaller learning rates.

Adagrad's performance remains consistent across different levels of sparsity (e.g., 20%, 50%, 80%) due to its **adaptive nature**, whereas other optimizers struggle with the imbalance between sparse and dense regions.

Codes link

Viswanath1680/
TAO-project

TAO project - CNN with Structured Sparsity

1 Contributor
0 Issues
0 Stars
0 Forks

GitHub

GitHub - Viswanath1680/TAO-project: TAO project - CNN with Structur...

TAO project - CNN with Structured Sparsity. Contribute to Viswanath1680/TAO-project development by creating an account on GitHub.

Conclusion

This extension successfully explored the interplay between sparsity and various optimization algorithms in CNNs for image classification tasks. **Our experiments demonstrated that structured sparsity can significantly reduce model size** and computational requirements with minimal loss in accuracy, especially on simpler datasets like CIFAR-10.

Adagrad emerged as a superior optimizer for training sparse models, balancing training time and accuracy effectively. However, the benefits of sparsity were more pronounced in CIFAR-10 compared to CIFAR-100, highlighting the influence of dataset complexity on sparsity's effectiveness.

THANK YOU