Optimizing Image Classification Accuracy through MWU on Neural Networks

Yixiao Chen YCHEN3@UNC.EDU
Winni Yong-Yu Yu WYU3@UNC.EDU
Anmeng Cao ANMENG@AD.UNC.EDU
Tianjun Ma MOSKEN@AD.UNC.EDU

1. INTRODUCTION

Image classification has many uses in real-world applications ranging from autonomous driving to content categorization. The CIFAR-10 dataset, known for its diversity across ten object categories, provides a challenging yet tractable benchmark for image classification. In this project, we explore the application of an ensemble-based approach to image recognition using the CIFAR-10 dataset. By combining predictions from four simple convolutional neural networks (CNNs) through the Multiplicative Weights Update (MWU) algorithm, we aim to enhance classification accuracy. We trained the four models on the CIFAR-10 training set and utilized a portion of the test set to fine-tune the weights used in our ensemble system. The project investigates various loss functions, weight update methods, and prediction strategies to effectively aggregate the predictions from the four models and make a robust final decision. Our results show that using the MWU algorithm for ensemble learning can improve image classification performance. Our code is available at https://github.com/cyixiao/CNN-MWU-classification.

2. DATASET

2.1. *Description*

The CIFAR-10 dataset is a subset of the much larger "80 million tiny images dataset", assembled by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. This is one of the most widely used datasets for machine learning research. It comprises 60,000 color images of 32x32 resolution, uniformly distributed across ten distinct classes, making it a standard benchmark in machine learning for image recognition tasks. The ten classes are as follows: Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck. CIFAR-10 is structured into six batches, where each batch contains 10,000 images. Specifically, the dataset is split into five training batches and one test batch, ensuring comprehensive coverage for model training and evaluation.

2.2. *Usage*

For our models, we first used the training batches to train CNNs, and the test batch to evaluate their accuracy. Next, to apply the Multiplicative Weights Update (MWU) algorithm in building our ensemble system, we utilized the test batch differently. We selected 1,000 images from the test set to run the MWU algorithm and adjust the weights for each model in our system. This limited subset prevents the weights from growing excessively. Once the weights were trained, we made predictions on the remaining 9,000 images from the test batch to evaluate the overall accuracy of our system.

3. METHODOLOGY

3.1. *Multiplicative Weights Update*

The Multiplicative Weights Update (MWU) algorithm is an approach for online learning and decision-making that combines predictions from a set of experts using a weighted average. After each prediction round, the algorithm modifies the weights of the experts based on their predictive performance, increasing the weights for accurate predictions and decreasing them for less accurate predictions.

3.2. *Experts*

Our four experts are four widely cited CNNs used for CIFAR-10 image recognition. They include traditional architectures with alternating convolutional and pooling layers (see Figure 1), as well as VGG-like structures with consecutive convolutional layers followed by a pooling layer (see Figure 2). Each of these networks varies in depth, activation functions, and regularization methods, resulting in different performance levels. Each model was independently trained on the 50,000 images from the training set.

3.3. *Loss Function*

In this project, we explored six different loss functions for the MWU algorithm. Each loss function guides the weight adjustment process differently based on the performance of each expert model.

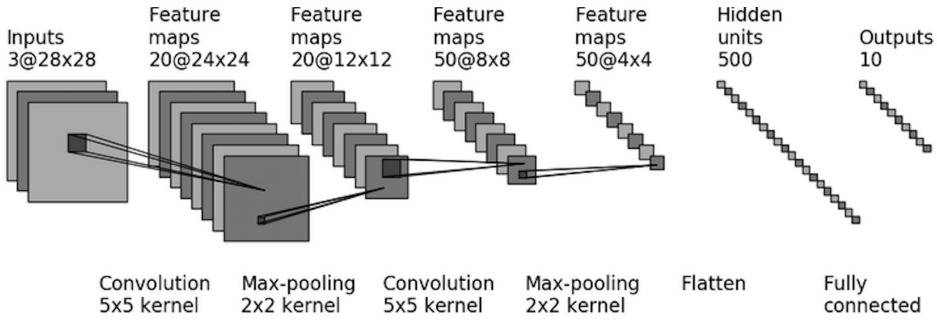

FIG. 1.— Schematic of the CNN architecture with alternating convolutional and pooling layers.

3.3.1. *Symmetrical Reward-Penalty Loss*

This loss function makes the reward for a correct prediction equivalent to the penalty for an incorrect prediction.

$$L(i) = \begin{cases} -1 & \text{if the prediction is correct} \\ 1 & \text{if the prediction is incorrect} \end{cases}$$

3.3.2. *Asymmetrical Penalty-Heavy Loss*

This loss emphasizes penalizing incorrect predictions more heavily than rewarding correct ones.

$$L(i) = \begin{cases} -0.5 & \text{if the prediction is correct} \\ 1 & \text{if the prediction is incorrect} \end{cases}$$

3.3.3. *Asymmetrical Reward-Heavy Loss*

This loss emphasizes rewarding correct predictions more heavily than penalizing incorrect ones.

$$L(i) = \begin{cases} -1 & \text{if the prediction is correct} \\ 0.5 & \text{if the prediction is incorrect} \end{cases}$$

3.3.4. *Weighted Majority Algorithm (WMA) Loss*

This approach resembles the WMA, with no reward for correct predictions.

$$L(i) = \begin{cases} 0 & \text{if the prediction is correct} \\ 1 & \text{if the prediction is incorrect} \end{cases}$$

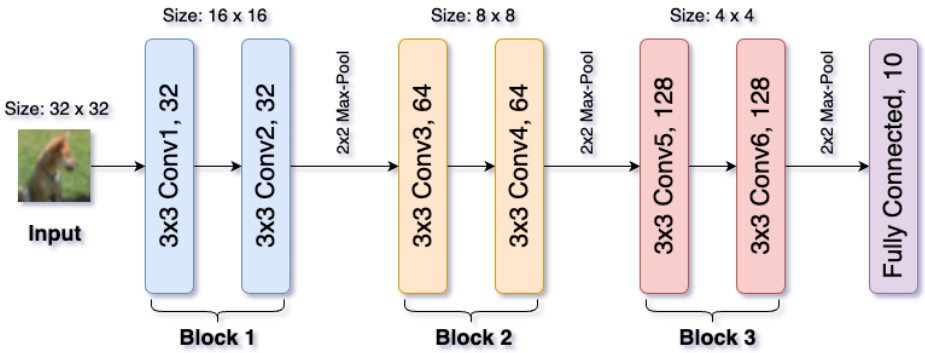

FIG. 2.— Schematic of the VGG-like CNN architecture with consecutive convolutional layers followed by a pooling layer.

3.3.5. *Cross-Entropy Loss*

This loss function is widely used in classification tasks because it can effectively handle probability distribution.

$$L(i) = -\log\left(\frac{\exp(o_{i,y})}{\sum_j \exp(o_{i,j})}\right)$$

3.3.6. *Mean Squared Error (MSE) Loss*

Although MSE is traditionally used in regression tasks, we adapted it for classification by converting the target labels to one-hot vectors. This loss function computes the squared differences between the predicted and actual probabilities.

$$L(i) = \frac{1}{C}\sum_{j=1}^{C}(\sigma(o_{i,j}) - t_j)^2$$

3.4. *Weight Update Methods*

We also used four different weight update methods to adjust the weights of each expert model in the MWU algorithm.

3.4.1. *Traditional MWU Weight Update*

This method modifies the weight of each expert based on the normalized loss.

$$w[i] = w[i] \cdot (1 - \epsilon \cdot \frac{L(i)}{\rho})$$

3.4.2. *Exponential Weighting*

This approach exponentially reduces the weight of underperforming experts and slightly increasing the weight of more accurate ones.

$$w[i] = w[i] \cdot \exp(-\epsilon \cdot L(i))$$

3.4.3. *Additive Weighting*

In this method, weights are updated by directly add a reward or penalty to each expert's weight.

$$w[i] = \max(w[i] + \epsilon \cdot (\rho - L(i)), 0)$$

3.4.4. *Adaptive Epsilon*

This one dynamically adjusts the epsilon to improve the stability of the algorithm, then followed by the Exponential Weighting.

$$\epsilon_i = \frac{\epsilon}{\sqrt{k+1}}$$

The weights are updated using:

$$w[i] = w[i] \cdot \exp(-\epsilon_i \cdot L(i))$$

3.5. *Prediction Strategies*

We used three different Prediction Strategies to predict the types of images using the trained weight and calculate the accuracy.

3.5.1. *Random Selection Based on Normalized Weights*

In this strategy, the weights of all experts are first normalized to sum to one, then an expert is selected randomly based on this distribution (shown in the following formula), and the output of this selected expert becomes the system's final prediction. In this way, the experts with higher weights have a proportionally higher chance of being selected.

$$p[i] = \frac{w[i]}{\sum_j w[j]}$$

3.5.2. *Weighted Voting Based on Cumulative Weights*

In this approach, each expert votes for a class based on its prediction. Each class receives votes weighted by the trained weights, then the class with the highest cumulative weight across all experts is selected as the final prediction.

$$W(c) = \sum_{i=1}^{m} w[i] \cdot I(y[i] = c)$$

The final prediction is:

$$\hat{y} = \arg\max_c W(c)$$

3.5.3. *Bayesian Model Averaging*

In this method, each expert provides a set of probability for each class. Then these probabilities are multiplied by the expert's weight, and the result are combined into a single set of probabilities. The class with the highest combined probability is selected as the final prediction.

$$P(c) = \sum_{i=1}^{m} w[i] \cdot p[i, c]$$

The final prediction is:

$$\hat{y} = \arg\max_c P(c)$$

4. RESULT AND DISCUSSION

4.1. *Loss Function*

In our experiments, the **Cross-Entropy Loss** consistently performed the worst across all combinations of weight update methods and prediction strategies. This is because its logarithmic nature strongly penalizes errors in the correct class, which reduces the system's generalization ability. Additionally, it may not differentiate model performances sufficiently, causing the weights to remain relatively static and unable to accurately represent performance.

All other loss functions showed similar results across most weight update methods and prediction strategies. However, in the **Random Selection Based on Normalized Weights** strategy, the **Asymmetrical Reward-Heavy Loss** and **Symmetrical Reward-Penalty Loss** (where a correct prediction is rewarded with -1) performed better than others, as shown in Figure 3. By giving a higher reward for correct predictions, these loss functions ensured that models with consistent accuracy maintained higher weights, making them more likely to be selected. However, in non-random strategies, the final prediction is directly influenced by the cumulative weights. In such cases, high-reward models mean small differences in rewards or penalties have a limited effect on the overall prediction, so these loss functions don't offer a significant advantage over others.

4.2. *Weight Update Methods*

For all loss functions, when using **Random Selection Based on Normalized Weights** and **Bayesian Model Averaging**, the **Additive Weighting** method consistently performed the worst, while the other weight update methods gave similar results. This is probably because the **Additive Weighting** method simply adds or subtracts a fixed value to the weights. This can cause erratic changes

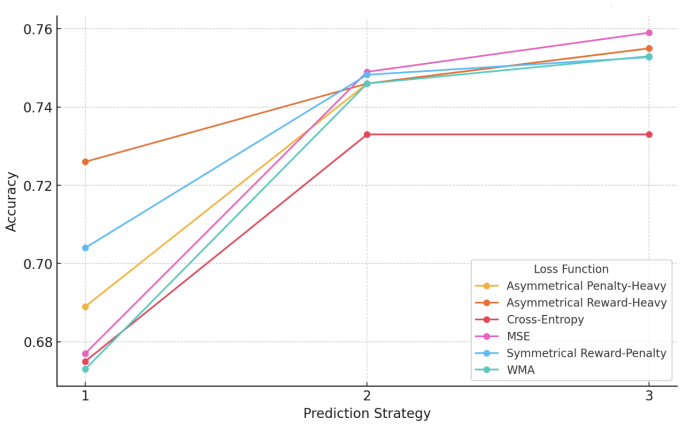

FIG. 3.— Accuracy by Loss Function Across Prediction Strategies. The graph compares the performance of various loss functions across three prediction strategies (1, 2, and 3), in the same order as introduced in the prediction strategy section. All experiments were conducted using the Exponential Weight Update method to compare the loss function performance across prediction strategies.

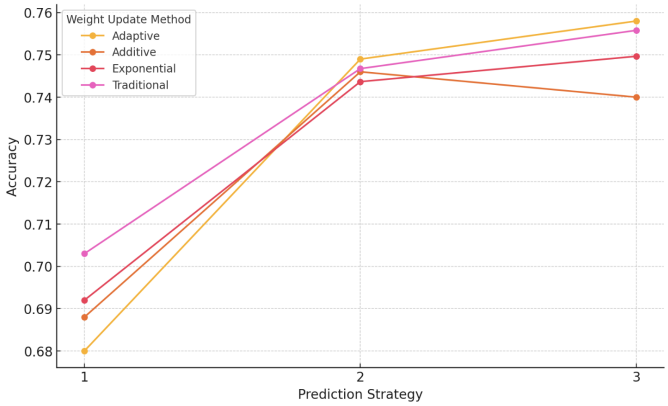

FIG. 4.— Accuracy by Weight Update Methods Across Prediction Strategies. The graph compares the performance of various weight update methods across three prediction strategies (1, 2, and 3), in the same order as introduced in the prediction strategy section. All experiments were conducted using the MSE Loss to compare the weight update methods performance across prediction strategies.

and overly penalize underperforming models, leading to inaccurate final predictions.

However, when using **Weighted Voting Based on Cumulative Weights**, all weight update methods performed similarly, including the **Additive Weighting** method (see Figure 4). This is because the final prediction directly relies on the cumulative weights, not the probability, and even small weight changes can still influence the final decision. As a result, variations in weight adjustments between different methods do not significantly impact the overall performance of this strategy.

4.3. *Prediction Strategy*

In our experiments, **Bayesian Model Averaging** always performed better than **Random Selection Based on Normalized Weights** and **Weighted Voting Based on Cumulative Weights** across all combinations of loss functions and weight update methods. This is because **Bayesian Model Averaging** combines the probabilities from all experts, using their weights to create a final prediction. This way, the final decision considers the nuanced contributions of each expert, leading to more accurate predictions.

On the other hand, **Random Selection Based on Normalized Weights** only picks one expert based on their weight probability, ignoring the predictions of other experts. This leads to the worst performance as it can't effectively use the combined output of all experts. Additionally, the randomness adds uncertainty, which means that even if the best-performing expert has a high weight, the model might still select the worst-performing expert, increasing the error. **Weighted Voting Based on Cumulative Weights** does better but is still limited because it depends only on the votes from experts

TABLE 1
Final Prediction Accuracy Comparison

Model	Accuracy
Expert 1	65%
Expert 2	73%
Expert 3	71%
Expert 4	60%
WMU System	76%

with higher weights, possibly ignoring valuable predictions from less dominant experts. Thus, it doesn't match the comprehensive combination provided by **Bayesian Model Averaging**.

4.4. *General Discussion*

By experimenting with all combinations of loss functions, weight update methods, and prediction strategies, we found that the best combination is the **MSE Loss** function with any weight update method (although **Additive Weighting** is just slightly worse) and using **Bayesian Model Averaging**. This combination achieved an accuracy of 76%, which is 3% higher than the best individual expert's accuracy of 73%, as shown in Table 1. This result demonstrates that our approach can leverage an ensemble system to improve model predictions effectively.

5. CONCLUSION AND FUTURE WORK

In this project, we explored an ensemble learning approach using the Multiplicative Weights Update (MWU) algorithm to improve image classification on the CIFAR-10 dataset. By combining predictions from four convolutional neural networks (CNNs) through various loss functions, weight update methods, and prediction strategies, we demonstrated the potential of an ensemble system to significantly enhance model predictions compared to individual models.

In future work, we aim to improve the CNN experts to work with more robust models, introduce additional weight update algorithms, loss functions, and prediction strategies, and test our method on a broader range of datasets.