Cifar-10 Classification with Three-layer Neural Networks

Zijie Qiu
Department of Big Data
Fudan University
`22307140109@m.fudan.edu.cn`

Abstract

This paper presents a comprehensive investigation into the classification of the CIFAR-10 dataset using a three-layer neural network implemented from scratch with NumPy. We explore key hyperparameters including hidden layer dimensions, learning rate, regularization strength, and activation functions. Through systematic experimentation, we established an optimal configuration with hidden layer sizes of 2048, a learning rate of 0.05, L2 regularization strength of 0.001, and ReLU activation. Our final model achieves 57.6% accuracy on the test set, demonstrating the capabilities and limitations of shallow neural networks on complex image classification tasks. We provide detailed ablation studies and visualizations of learned features to offer insights into the network's behavior. This work serves as both an educational exploration of fundamental neural network principles and a baseline for more sophisticated approaches to image classification. Implementation for this project are availbale on github [1], and the dataset and model weights are provided on google drive [2].

1 Dataset Introduction

The CIFAR-10 dataset Krizhevsky and Hinton [2009] is a well-established benchmark in computer vision and machine learning research, consisting of 60,000 32×32 color images distributed across 10 distinct classes. These classes encompass common objects such as airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks, with 6,000 images per class. The dataset is officially partitioned into 50,000 training images and 10,000 test images.

Each image in CIFAR-10 is represented as a three-channel RGB image with a low resolution of 32×32 pixels, making it computationally manageable while still presenting sufficient complexity for meaningful algorithm development and evaluation. Despite its relatively small image size compared to modern datasets, CIFAR-10 continues to serve as an important benchmark due to its balanced class distribution, manageable scale, and challenging nature - the small image size and intra-class variation make perfect classification non-trivial even for sophisticated models.

For our experiments, we adhered to the default train-test split provided by the dataset, but further allocated 10% of the training data as a validation set for hyperparameter tuning and model selection. We applied normalization to all data splits using channel-wise statistics calculated from the training set. Specifically, we computed the mean (125.3414, 123.02483, 113.93862) and standard deviation (62.997307, 62.073704, 66.71649) across the three RGB channels of the training images. The normalization was then applied uniformly across all datasets using:

[1]https://github.com/Fatemoisted/Fudan-Vision-Lab1

[2]https://drive.google.com/drive/folders/196FOxoZPwiWhffu_uBOljlnB7ZrBzFvX?usp=sharing

$$X_{normalized} = \frac{X - \mu}{\sigma} \tag{1}$$

where X represents the original pixel values, while μ and σ denote the channel-wise mean and standard deviation vectors respectively. This normalization strategy ensures that each channel of the input images has approximately zero mean and unit variance, which typically improves the convergence behavior of neural networks during training.

2 Model Design

Our approach to CIFAR-10 image classification employs a three-layer neural network implemented from scratch without relying on deep learning frameworks like PyTorch. This implementation choice allows for a deeper understanding of the fundamental mechanics of neural networks while providing complete control over the optimization process.

2.1 Architecture Overview

The neural network consists of three fully connected layers with the following structure:

- **Input Layer:** The 32×32×3 RGB images are flattened into 3,072-dimensional vectors.
- **First Hidden Layer:** Transforms the input into an intermediate representation of dimension `hidden_size1`.
- **Second Hidden Layer:** Further refines the representation to dimension `hidden_size2`.
- **Output Layer:** Produces 10-dimensional logits corresponding to the CIFAR-10 classes, followed by softmax normalization.

The model supports both ReLU and sigmoid activation functions between layers, configurable during initialization.

2.2 Mathematical Formulation

Our three-layer neural network performs the following computations during forward propagation:

$$\mathbf{z}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)} \tag{2}$$

$$\mathbf{a}^{(1)} = \sigma(\mathbf{z}^{(1)}) \tag{3}$$

$$\mathbf{z}^{(2)} = \mathbf{a}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} \tag{4}$$

$$\mathbf{a}^{(2)} = \sigma(\mathbf{z}^{(2)}) \tag{5}$$

$$\mathbf{z}^{(3)} = \mathbf{a}^{(2)}\mathbf{W}^{(3)} + \mathbf{b}^{(3)} \tag{6}$$

$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z}^{(3)}) \tag{7}$$

where $\mathbf{X}$ is the batch of flattened input images, $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$ are the weight matrices and bias vectors, $\sigma(\cdot)$ is either ReLU or sigmoid activation, and $\hat{\mathbf{y}}$ contains the predicted class probabilities.

2.3 Objective Function

Our training objective consists of the cross-entropy loss with an L2 regularization term:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) + \frac{\lambda}{2}\sum_{l=1}^{3} ||\mathbf{W}^{(l)}||_F^2 \tag{8}$$

where N is the batch size, $C = 10$ is the number of classes, y_{ij} is the true label (one-hot encoded), $\hat{y}_{ij}$ is the predicted probability, λ is the regularization strength, and $||\mathbf{W}^{(l)}||_F^2$ is the squared Frobenius norm of the weights in layer l.

2.4 Implementation Details

Our implementation is written in pure NumPy, with all computations handled manually. The detailed pseudocode for our three-layer neural network implementation is provided in Algorithm 1 in Appendix A.

2.5 Activation Functions

For the activation function $\sigma(\cdot)$, we implement both options:

$$\text{ReLU}(x) = \max(0, x) \tag{9}$$

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

3 Methods

We trained a three-layer neural network on the training set for N_{epoch} epochs, validating performance after each epoch. The model with the highest performance on the validation dataset was saved, and its performance on the test set is reported. During training, we employed a learning rate decay strategy, multiplying the learning rate by a factor β_{lr} after each epoch.

To optimize model performance, we conducted an extensive hyperparameter search. Given the large number of tunable parameters—including hidden layer sizes (h_1 and h_2), learning rate (lr), L2 regularization strength (β_{reg}), learning rate decay ratio (β_{lr}), and batch size (B)—we divided them into groups by importance and performed a sequential search. The hyperparameter search space is defined in Table 1. We also consider the choice of the activation function (f_A) as a hyperparameter.

Table 1: Hyperparameter Search Space for Model Optimization

Hyperparameter	**Search Values**
h_1	256, 512, 1024, 2048
h_2	256, 512, 1024, 2048
lr	5×10^{-4}, 1×10^{-3}, 5×10^{-3}, 1×10^{-2}, 5×10^{-2}
β_{reg}	1×10^{-5}, 1×10^{-4}, 1×10^{-3}, 1×10^{-2}, 1×10^{-1}
β_{lr}	0.98, 0.95, 0.90, 0.85
B	64, 128, 256, 512
f_A	relu, sigmoid

Our search strategy consisted of two phases. First, we fixed β_{lr} at 0.95, B at 128, and N_{epoch} at 10 (to expedite the search process) while exploring all 400 possible combinations of h_1, h_2, lr, and β_{reg}. After identifying the optimal configuration for these four parameters, we sequentially optimized β_{lr}, B, and N_{epoch}. We set the activation fuction as relu for the expriments above and for the final parameter (f_A), we set N_{batch} sufficiently large to identify the point at which the model begins to overfit, thus to decide the best choice.

4 Experiment Results

4.1 Hyperparameter search results for h_1, h_2, lr, and β_{reg}

Our comprehensive search across the hyperparameter space identified an optimal configuration of $h_1 = 2048$, $h_2 = 2048$, $lr = 5 \times 10^{-2}$, and $\beta_{\text{reg}} = 1 \times 10^{-3}$. This configuration achieved a validation accuracy of 0.542. Detailed ablation studies for each of these hyperparameters are presented in the following figures.

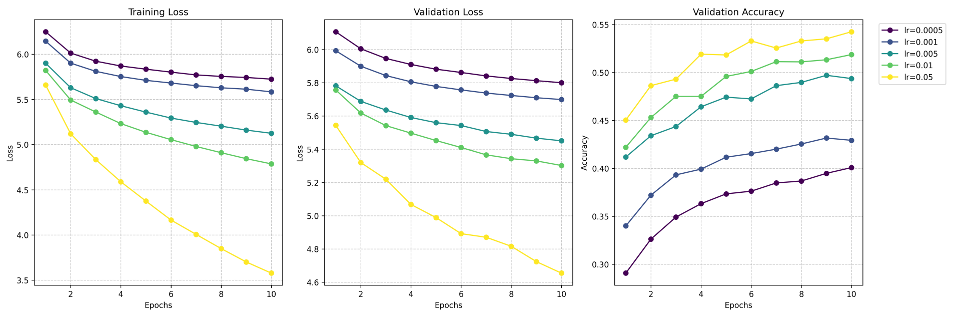

Figure 1: Metrics vs. Epochs for Different lr when h_1 is 2048, h_2 is 2048 and β_{reg} is 0.001

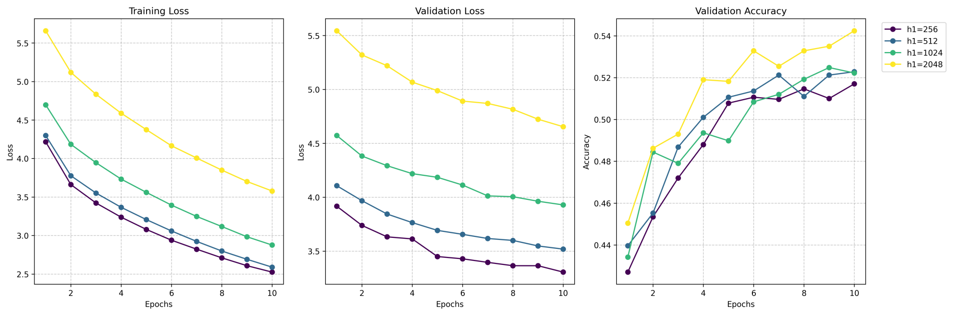

Figure 2: Metrics vs. Epochs for Different h_1 when lr is 0.05, h_2 is 2048 and β_{reg} is 0.001

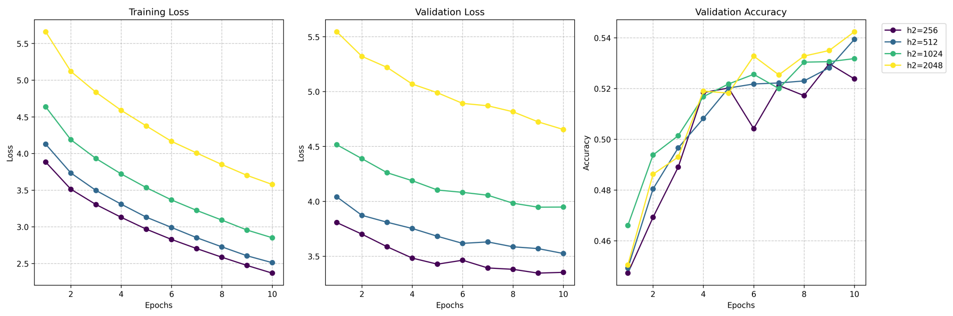

Figure 3: Metrics vs. Epochs for Different h_2 when lr is 0.05, h_1 is 2048 and β_{reg} is 0.001

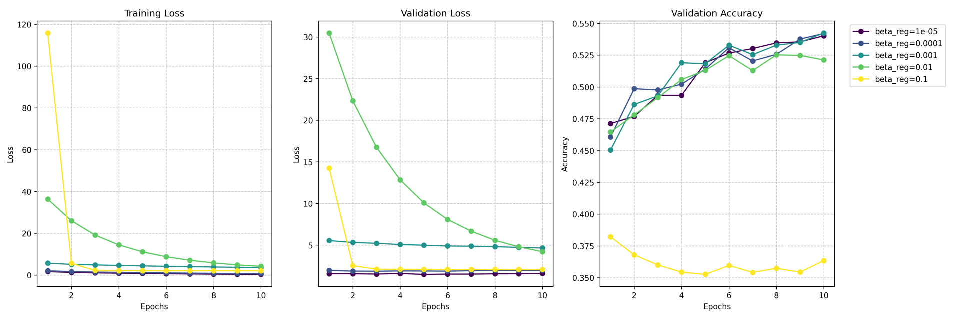

Figure 4: Metrics vs. Epochs for Different β_{reg} when lr is 0.05, h_1 is 2048 and h_2 is 2048

4.2 Hyperparameter Search Results for β_{lr}

After identifying the optimal values for lr, h_1, h_2, and β_{reg}, we conducted a targeted search to determine the most effective learning rate decay factor β_{lr}. We trained each model for 20 epochs. Figure 5 presents the performance metrics across training epochs for various β_{lr} values while maintaining the previously established optimal hyperparameters.

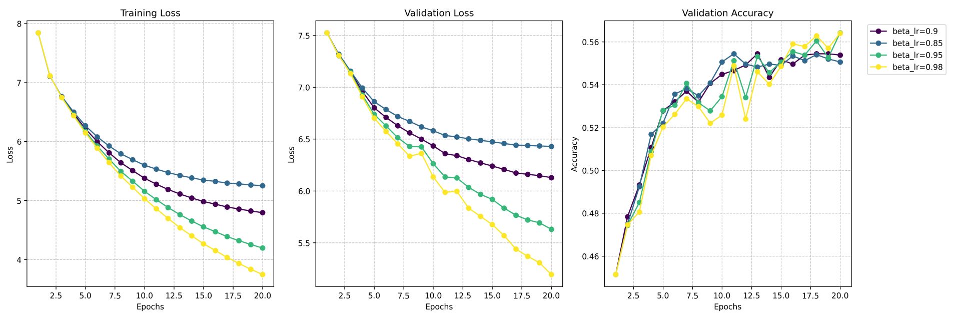

Figure 5: Metrics vs. Epochs for Different β_{lr} when lr is 0.05, h_1 is 2048, h_2 is 2048 and β_{reg} is 0.001

Our experiments reveal that setting $\beta_{lr} = 0.95$ yields the highest validation accuracy of 0.564, providing the optimal balance between learning rate decay and model convergence.

4.3 Hyperparameter Search Results for B

After identifying the optimal values for lr, h_1, h_2, β_{reg} and β_{lr}, we conducted a targeted search to determine the most effective batch size B. We trained each model for 20 epochs. Figure 5 presents the performance metrics across training epochs for various B values while maintaining the previously established optimal hyperparameters.

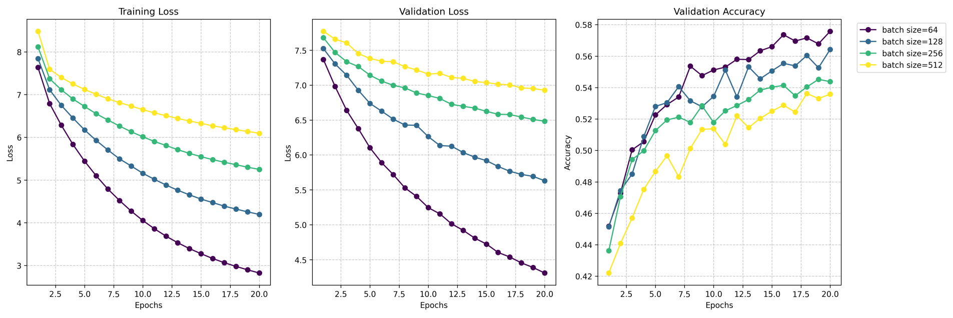

Figure 6: Metrics vs. Epochs for Different β_{lr} when lr is 0.05, h_1 is 2048, h_2 is 2048, β_{reg} is 0.001 and β_{lr} is 0.95.

Our experiments reveal that setting $B = 64$ yields the highest validation accuracy of 0.576, providing the optimal balance between learning rate decay and model convergence.

4.4 Activation Function Abalation

After determining the optimal hyperparameters, we conducted an ablation study to examine the impact of activation function choice on model performance. Using the established best hyperparameter combination, we trained the model with a sufficient number of epochs ($N_{train} = 40$) to ensure convergence. Figure 7 presents a comparative analysis of ReLU versus sigmoid activation functions.

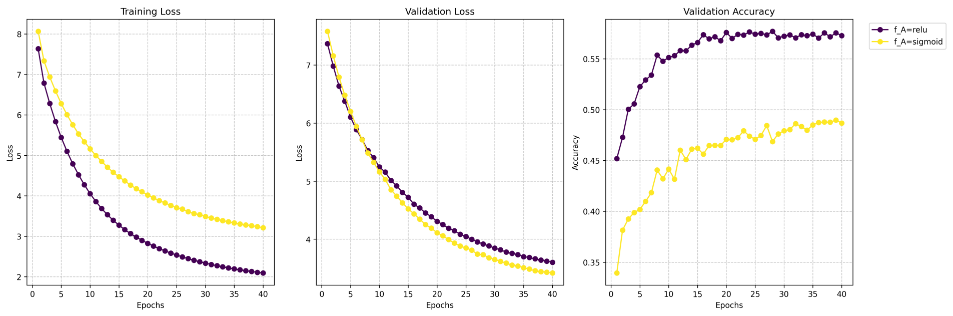

Figure 7: Metrics vs. Epochs for Different f_A when lr is 0.05, h_1 is 2048, h_2 is 2048, β_{reg} is 0.001, β_{lr} is 0.95 and B is 64.

The results demonstrate that ReLU significantly outperforms sigmoid as the activation function. The model with ReLU achieved a peak evaluation accuracy of 0.577 at epoch 28, whereas the sigmoid variant only reached 0.490 at epoch 39. Based on this substantial performance gap, we selected ReLU as the activation function for our final model architecture.

4.5 Final Performance on Testing Dataset

Our final model employs the following optimized hyperparameters: learning rate $lr = 0.05$, hidden layer dimensions $h_1 = h_2 = 2048$, regularization coefficient $\beta_{reg} = 0.001$, learning rate decay factor $\beta_{lr} = 0.95$, batch size $B = 64$, and ReLU activation function (f_A). Trained for 40 epochs, the model achieved its highest validation accuracy of 0.577 at epoch 28. When evaluated on the independent test dataset, it maintained consistent performance with an accuracy of 0.576.

5 Visualization of Neural Network Weights

To gain deeper insights into the learned features of our neural network model, we visualized the first layer weights. This visualization approach helps understand what patterns and features the network has learned to detect during training.

5.1 Weight Visualization Method

The first layer of our neural network directly connects to the input image pixels, with weights organized in a shape of $(3072, 2048)$, corresponding to connections between the $32 \times 32 \times 3$ input images and 2048 neurons in the first hidden layer. Each neuron essentially learns a template or filter that responds to specific visual patterns in the input space.

For visualization purposes, we reshaped each neuron's weight vector back to the original image dimensions ($32 \times 32 \times 3$) and normalized the weights within each color channel to enhance visibility. Specifically, for each neuron i and color channel c, we applied the following normalization:

$$W_{normalized}(i, x, y, c) = \frac{W(i, x, y, c) - \min_c(W)}{\max_c(W) - \min_c(W)} \tag{11}$$

where $W(i, x, y, c)$ represents the weight connecting pixel (x, y, c) to neuron i, and $\min_c(W)$ and $\max_c(W)$ are the minimum and maximum values within channel c for that neuron.

5.2 Interpretation of Weight Patterns

Figure 8 shows the visualization of weights for the first 16 neurons in the first hidden layer. These visualizations reveal several interesting patterns:

The visualized weights demonstrate that the network has learned to detect various low-level features such as edges, color transitions, and texture patterns. Neurons with strongly colored visualizations

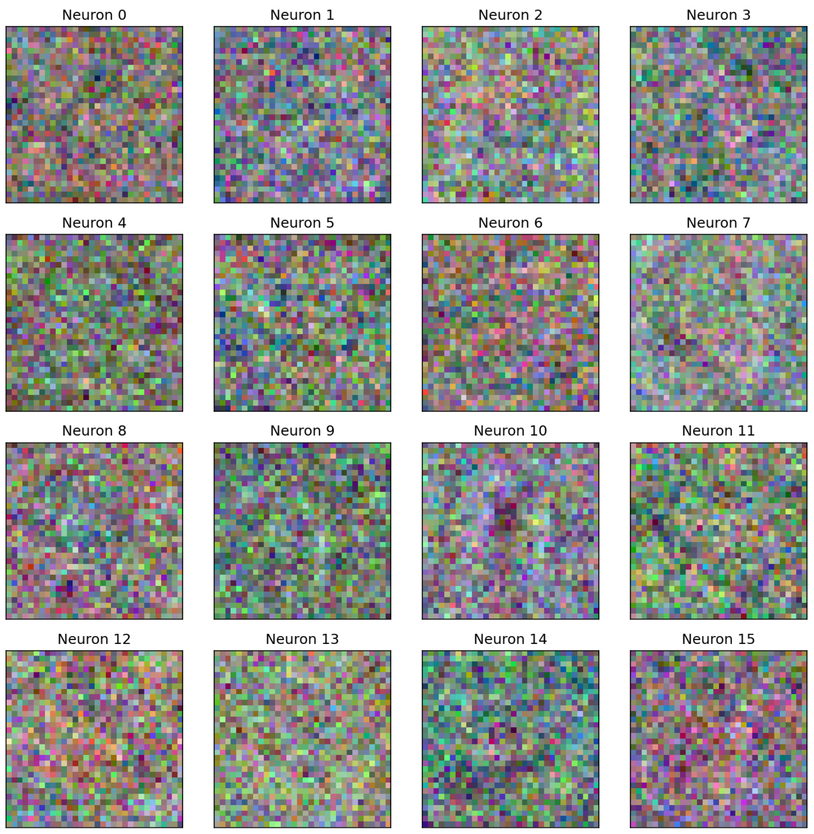

Figure 8: Visualization of the first layer weights for 16 neurons. Each image represents the weight pattern learned by an individual neuron, with color channels preserved to show the complete feature detector.

indicate specialization in detecting specific color features, while those with more grayscale-like patterns focus on structural elements regardless of color.

Some neurons (e.g., Neuron 7 and Neuron 10) appear to detect edge-like features with particular orientations, while others (e.g., Neuron 12) show more complex patterns that may correspond to corners or texture elements. This diversity of learned features forms the foundation for more complex feature detection in deeper layers of the network.

The color weight visualization reveals how the network processes each color channel differently, potentially highlighting the importance of color information for the classification task.

6 Discussion

This study presents a foundational exploration of three-layer neural networks for CIFAR-10 image classification. The achieved accuracy of 57.6% demonstrates both the capabilities and limitations of shallow networks when tackling complex image recognition tasks.

Several key observations emerge from our experimental results. First, network capacity proved crucial, with the largest hidden layer dimensions (2048 neurons) consistently outperforming smaller configurations. This suggests that for this dataset's complexity, representational capacity remains a limiting factor even with potential overfitting risks that were successfully mitigated through regularization.

The dramatic performance difference between ReLU and sigmoid activations (57.7% vs 49.0% validation accuracy) reinforces ReLU's effectiveness in modern neural architectures. ReLU's nonsaturating nature likely facilitated gradient flow throughout the network, enabling more effective learning of discriminative features.

The weight visualizations provide valuable insight into the feature detection mechanisms developed by the network. However, the relative simplicity of these patterns compared to those seen in deeper architectures may partially explain the performance ceiling encountered.

A key limitation of this approach is the fully-connected architecture, which ignores the spatial structure inherent in images. More advanced architectures like convolutional neural networks would likely achieve substantially higher performance by leveraging this structural information. Additionally, modern techniques such as batch normalization, dropout, and data augmentation could further improve results.

Future work could explore the transition point between shallow and deep architectures, investigating whether adding just one or two additional layers might yield disproportionate gains in performance. Additionally, examining the interaction between network depth and width could provide insights into efficient architectural design principles.

In conclusion, while the performance achieved is modest compared to state-of-the-art approaches, this investigation provides valuable educational insights into neural network fundamentals and establishes a solid baseline for understanding the progression toward more sophisticated architectures.

A Pseudocode Implementation

Below is the detailed pseudocode of our three-layer neural network implementation:

Algorithm 1 Three-Layer Neural Network Implementation

```
 1: Init: $W^{(i)} \sim \mathcal{N}(0, \sqrt{2/n_{i-1}})$, $b^{(i)} = 0 \; \forall i \in \{1, 2, 3\}$
 2: function FORWARD($X$)
 3:     for $i = 1 : 3$ do
 4:         $z^{(i)} = a^{(i-1)} W^{(i)} + b^{(i)}$ where $a^{(0)} = X$
 5:         $a^{(i)} = \begin{cases} \sigma(z^{(i)}) & i < 3 \\ \text{softmax}(z^{(i)}) & i = 3 \end{cases}$
 6:     end for
 7:     cache values, return $a^{(3)}$
 8: end function
 9: function BACKWARD($y, \lambda$)
10:     $dz^{(3)} = a^{(3)} - y$
11:     for $i = 3 : -1 : 1$ do
12:         $dW^{(i)} = a^{(i-1)T} dz^{(i)} + \lambda W^{(i)}$
13:         $db^{(i)} = \sum_j dz_j^{(i)}$
14:         if $i > 1$ then
15:             $da^{(i-1)} = dz^{(i)} W^{(i)T}$
16:             $dz^{(i-1)} = da^{(i-1)} \odot \sigma'(z^{(i-1)})$
17:         end if
18:     end for
19:     return grads
20: end function
```

References

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.