Rookie: A Three-layer Neural Network for Image Classification*

Yuanye Liu
School of Data Science
Fudan University
yuanyeliu@fudan.edu.cn

Abstract

Deep learning has achieved remarkable success across a wide range of tasks. However, the majority of implementations rely on established frameworks such as PyTorch and TensorFlow. To gain a deeper understanding of the fundamental principles of deep learning, such as backward propagation, manual implementation is crucial. In this work, we introduce Rookie, a simple three-layer neural network, manually implemented using NumPy to illustrate the core mechanics of neural operations. We evaluated Rookie on the Fashion-MNIST dataset, where it demonstrated commendable performance, achieving an accuracy of 89.84% on the test set. The source code and trained model weights are accessible at https://github.com/HenryLau7/ROOKIE/ and google drive, respectively.

1 Introuduciton

In the recent decade, deep learning has shown huge potential on many fields. However, the core technique neural netwrok has a rich history spanning several decades. Due to the efforts of a dedicated community of scientists, python frameworks sunch as PyTorch, TensorFlow, and Caffe have significantly simplified the process of developing neural networks.

Nevertheless, a fundamental understanding of neural network mechanisms is essential for deeper insights into their functionality and potential. Constructing a neural network from the ground up, without the help of advanced libraries, becomes an invaluable exercise. This project aims to build a basic three-layer neural network using only Numpy for mathematical operations, data handling and parameter management. Through this approach, we manange to enhance a deeper understanding of neural network operations and principles. We then evaluate its effectiveness on Fashion-MNIST[1], gaining a remarkable results on image classification.

The outline of this report is as follows. In section 2, we will introduce the model structure and the process of forward computation and backward propagation. In addition, SGD with minibatch will be introduced in this section. In section 3, we will introduce and analyze experimental results on Fashion-MNIST. In section 4, We will visualize the parameters of this model to explore the pattern recognition of neural network. In section 5, we will give a concise conclusion.

2 Method

In this study, we propose a three-layer neural network, termed **Rookie**, specifically designed for the task of image classification. As depicted in Fig.1, the network receives image data accompanied by corresponding labels as input. It comprises three hidden layers that linearly transform the features. To introduce non-linearity into the model, a ReLU (Rectified Linear Unit) activation function is em-

*Course project of DATA620004: Neural Network and Deep Learning

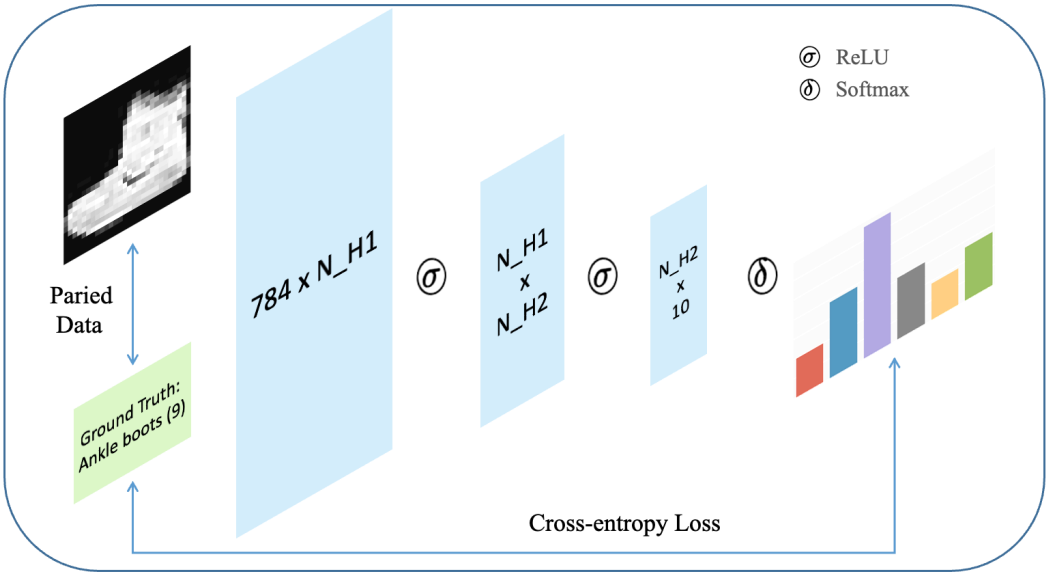


Figure 1: Framework of Rookie.

ployed. For the classification task, a Softmax function is utilized to generate multinomial probability distributions over the various class labels.

2.1 Forward Computing in Neural Networks

The process of forward computing in neural networks, particularly for image classification tasks, involves transforming input images $\boldsymbol{x} \in \mathbb{R}^{\text{Din}}$ with a corresponding ground truth label y, through a series of computational layers to extract relevant features. This transformation is facilitated by three linear layers, each followed by a non-linear activation function. The computation at the i_{th} layer can be formally described as Eq. 1,

$$\begin{aligned} z_i &= W_i \cdot a_{i-1} + b_i, \\ a_i &= \text{ReLU}(z_i), \end{aligned} \tag{1}$$

where z_i represents the linear transformation of the activation from the previous layer (a_{i-1}), W_i denotes the weight matrix, b_i signifies the bias, and **ReLU** introduces a non-linear operation essential for learning complex patterns. The **ReLU** function is defined as,

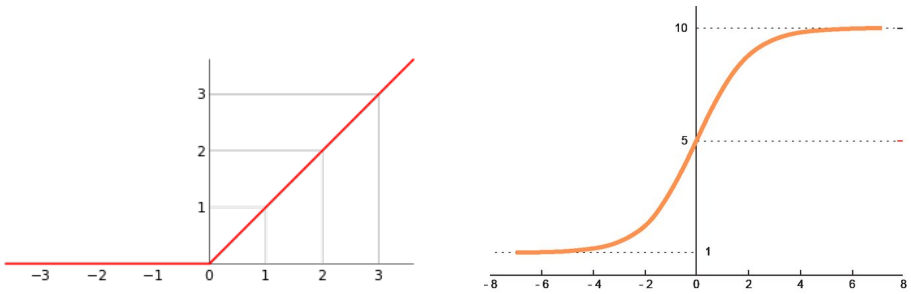
$$\text{ReLU}(x) = x^+ = \max(0, x) = \frac{x + |x|}{2} = \begin{cases} x & \text{if } x > 0, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

This non-linear activation plays a critical role in enhancing the network's capability to capture intricate data representations. The dimensionality of the weights and biases across the three layers are specifically configured as $W_1 \in \mathbb{R}^{\text{Din}\times\text{H1}}, W_2 \in \mathbb{R}^{\text{H1}\times\text{H2}}, W_3 \in \mathbb{R}^{\text{H2}\times K}$; and $b_1 \in \mathbb{R}^{\text{H1}\times 1}, b_2 \in \mathbb{R}^{\text{H2}\times 1}, b_3 \in \mathbb{R}^{K\times 1}$, respectively, where K is the total classes.

Upon reaching the final layer, the model computes the logits which are then transformed into a multinomial distribution $\boldsymbol{p} = [p_1, \ldots, p_K]$ through the softmax function, facilitating the classification task. This transformation is mathematically represented as:

$$\hat{p_i} = \text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad \text{for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K. \tag{3}$$

The concrete form of ReLU and softmax is illustrated in Fig. 2.1



(a)ReLU (b)Softmax

Figure 2: Activation function.

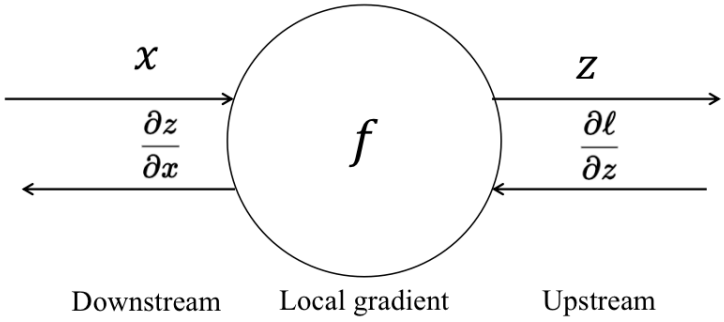


Figure 3: Computation graph of backpropagation

In the final step of the forward computing process, the model generates a prediction by selecting the class with the highest probability from the output distribution. This is mathematically represented as $\hat{y} = \arg\max_i p_i$, where $\hat{y}$ is the predicted label corresponding to the highest probability p_i among the computed probabilities in the vector $\boldsymbol{p}$. This step effectively concludes the forward pass of the network, setting the stage for subsequent backward propagation and optimization phases

2.2 Backward propagation

Gradient descent is a widely utilized method for identifying the optimal solution in optimization problems. The objective is to determine the most fitting network parameters W_i, b_i that accurately represent the data. In the context of deep learning, the chain rule is employed to calculate the gradient of each network parameter, a process depicted in Figure 2.2.

For the linear layers within our model, described by the equation $z = Wx + b$, the gradient computation unfolds as follows:

$$\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial z} \cdot \frac{\partial z}{\partial W} = \frac{\partial \ell}{\partial z} \cdot x, \tag{4}$$

where ℓ denotes the loss, and x signifies the input to the layer, serving as an intermediate variable in the network's computations. During practical implementation, it is common to store (or cache) the results of forward computations to efficiently retrieve them during the backward propagation phase. This strategy optimizes the computational resource usage and expedites the gradient calculation process across the network.

For activation we used, *ReLu, Softmax(with cross-entropy loss)*, the backward propagation is,

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}x} ReLU(x) &= 1\{x > 0\}, \\ \frac{\mathrm{d}}{\mathrm{d}x} \ell(Softmax(x), y) &= Softmax(x) - y. \end{aligned} \tag{5}$$

Table 1: The available parameters for tuning.

Parameters	Values
Learning rate	$0.5, 0.4, 0.3, 0.2, 0.1, 0.05$
Reg λ	$1e-3, 1e-4, 1e-5$
Hidden layer 1	$500, 400$
Hidden layer 2	$324, 256$
LR decay	$0.8, 0.5, 0.3$

2.3 Stochastic Gradient Descent on Minibatches

Stochastic Gradient Descent (SGD) stands as a cornerstone optimization technique in the realm of machine learning and deep learning. Distinguished from the traditional Gradient Descent (GD) approach, which computes the gradient of the loss function with respect to the parameters across the entire dataset, SGD introduces a more computationally efficient alternative. Instead of utilizing the full dataset for each iteration, SGD selects a minibatcha small, randomly chosen subset of the datato compute the local loss and its corresponding gradient. This process is formally expressed in the algorithm SGD, which iteratively updates the model parameters based on the gradients calculated from these minibatches. The primary advantage of SGD lies in its improved computational efficiency and faster convergence rates, especially when dealing with large datasets. By operating on minibatches, SGD reduces the memory requirements and accelerates the gradient computation process, allowing for more frequent updates to the model parameters within the same computational budget as traditional GD.

In this study, we divided the dataset into several data batches, and utilize SGD in each mini-batch.

2.4 Training Paradigm

In the training of neural network models, particularly for classification tasks, cross-entropy serves as a popular loss function. This criterion is especially effective in measuring the discrepancy between the predicted probability distribution $\boldsymbol{p}$ over classes and the actual distribution represented by the one-hot encoded label $\tilde{\boldsymbol{y}} = \text{one-hot}(y)$. The computation of cross-entropy loss for a single instance can be articulated as follows:

$$H(\tilde{\boldsymbol{y}}, \boldsymbol{p}) = -\sum_{i=1}^{K} \tilde{y}_i \log(p_i), \tag{6}$$

where K is the number of classes, $\tilde{y}_i \in \{0, 1\}$ is the i^{th} component of the one-hot encoded true label vector $\tilde{\boldsymbol{y}}$, and p_i is the i^{th} component of the predicted probability vector $\boldsymbol{p}$ for the corresponding class. This loss function is minimized when the predicted probability distribution closely matches the true distribution, effectively guiding the model towards more accurate predictions.

During the training process, the cross-entropy loss is computed for each instance in the dataset, and the model parameters are updated iteratively using an optimization algorithm SGD. By minimizing this loss during training, we aim to adjust the model parameters (W_i, b_i) to reduce the divergence between the model's predictions and the actual labels, thereby enhancing the model's classification accuracy.

3 Experiment

3.1 Dataset

Our model's effectiveness was validated using the Fashion-MNIST dataset [1], one of the most popular datasets within the machine learning community. Fashion-MNIST is a dataset commonly used in machine learning and computer vision for benchmarking image classification algorithms. Developed as an alternative to the traditional MNIST datasetwhich consists of handwritten digits. The dataset contains $70,000$ grayscale images, divided into a standard split of $60,000$ training and

Table 2: The top 10 results on Fashion-MNIST with different hyper-parameters combination. Rookie achieved a best performance of 89.84%.

Learning rate	λ	Hidden layer 1	Hidden layer 2	LR decay	Test Accuracy ↑
0.2	0.001	500	324	0.5	**89.84%**
0.3	0.001	400	324	0.2	89.81%
0.1	0.001	500	324	0.5	89.79%
0.3	0.001	400	256	0.2	89.78%
0.4	0.001	400	324	0.2	89.78%
0.2	0.001	400	256	0.2	89.75%
0.3	0.001	400	256	0.5	89.74%
0.2	0.001	500	324	0.2	89.7%
0.2	0.001	500	256	0.2	89.7%
0.3	0.001	500	324	0.2	89.68%

$10,000$ testing images. Each image is a 28×28 pixel representation of a fashion item, ranging from 0 to 9, providing a total of 10 distinct classes for classification tasks. The simplicity of the dataset, combined with its reasonably sized dimensionality, makes Fashion-MNIST an ideal benchmark for assessing the performance of various image recognition algorithms and neural network architectures.

We randomly select 5000 samples from training set to form validation set. For the whole dataset, we utilize z-score normalization. Before each epoch, we also employ batch normalization for each mini-batch. Notably, we do not introduce batch normalization layer in the network to learn and update, for simplicity we only use it as a pre-process.

3.2 Implementation detail

The model is structured to include two hidden layers, with the first and second layers consisting of 500 and 324 nodes, respectively. Batchsize was set as 512. The weight matrices W_is are initialized following a zero-mean Gaussian distribution with a variance of $\sqrt{2/\#\text{node}}$, where $\#$node refers to the number of nodes in the respective layer. The bias b_is are initialize as zeros. We trained the model for 500 epochs in total. The implementation was employed with Numpy. All experiments are conducted on Apple M3 Pro CPU. As shown in Table 1, we choose learning rate, regularization coefficient λ and weight decay for leanring rate as tuning parameters.

3.3 Results

We conduct several experiments to find the most suitable parameters combination. As illustrated in Table 2, the best performance on Fashion-MNIST achieved a 91.38% accuracy on validation set and an 89.84% accuracy on test set. In several combinations, the one with leanring rate 0.2, regularization parameter $\lambda, 1e-3$ and Hidden layer $[500, 324]$, learning rate decay 0.5 achieved the best performance.

Notably, the first three rows are ablation study for the regularization and weight decay for learning rate. The regularization is essential, which can avoid the over-fitting to some extent. In addition, the weight decay for learning rate can assist to converge to the minima, especially in a local miniaml area.

In addition, we plotted the loss and accuracy curves, as illustrated in Fig. 3.3.

4 Visual study

In this section, we conducted visualization for the network parameters, to explore the pattern recognition of neural network. Specially, we visualize $W1, W3$ and the complete transform $W = W_1 \cdot W_2 \cdot W_3$.

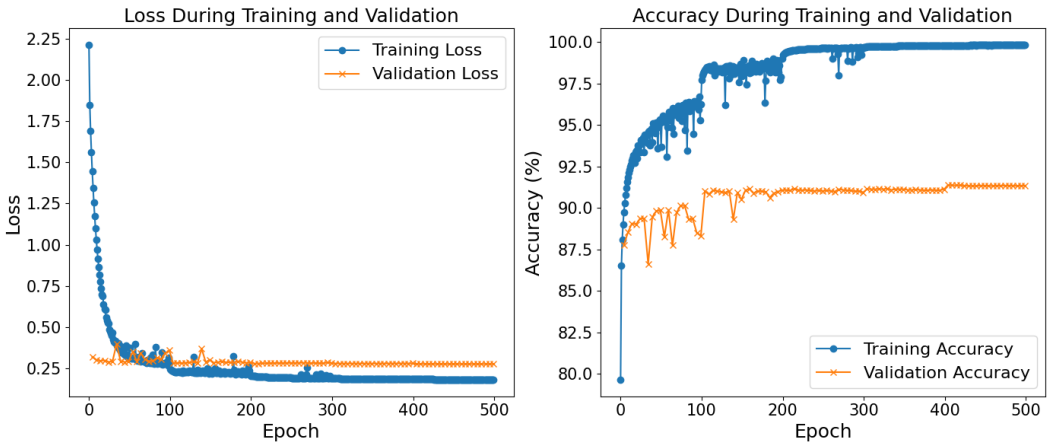


Figure 4: Visualization of loss and accuracy on training set and validation set.

4.1 Visualization of W_1

The first transformation $W_1 \in \mathbb{R}^{784\times500}$, which is to learn 500 featuer of the source images. Here we visualize as a 28×28 image for each features, as shown in the Fig. 4.1.

From the Fig. 4.1, we can find the network is learning some basic features in the very first layer, such as the circle, line and edges.

To further study the significant features, we employed Principal Component Analysis (PCA) techniques, to compresses to the most important 20 featuers.

Fig. 4.1 expresses the 20 most significant features in the first layer, to distinguish different fashion items. Specially, it shows more complex element like specific strokes than the Fig. 4.1.

4.2 Visualization of W_3

The last layer $W_3 \in \mathbb{R}^{300\times10}$, dirrectly map the feature to the classification. As Fig. 4.1 shows, however, it's hard to explore the comprehensible patterns for humans. That's because the this transformation expresses the linear projection of 300 features that extracted by former layers. These features are hard to understand themselves. In addition, the resolution is relatively lower, which can be one of the reason.

4.3 Visualization of combined transformation W

To validate the network can learn the pattern of each fashion item,we visualized the combined $W = W_1 \cdot W_2 \cdot W_3$, *i.e.*, $W \in \mathbb{R}^{784}$.

As illustrated in Fig. 4.1, we can find the W completely learn the pattern of each fashion item, clear edge and shadow. Notably, we didn't consider the non-linear activation and the bis b, which consist of critical information about the pattern.

5 Conclusion

In this study, we proposed a simple three-layer neural network, named Rookie. Specifically, we use Numpy to manually implement the forward computation and backward propagation. Experimental results shows satisfactory performance on Fashion-MNIST. Additionally, we conduct visual justification for each layer to explore the pattern recognition of neural network.

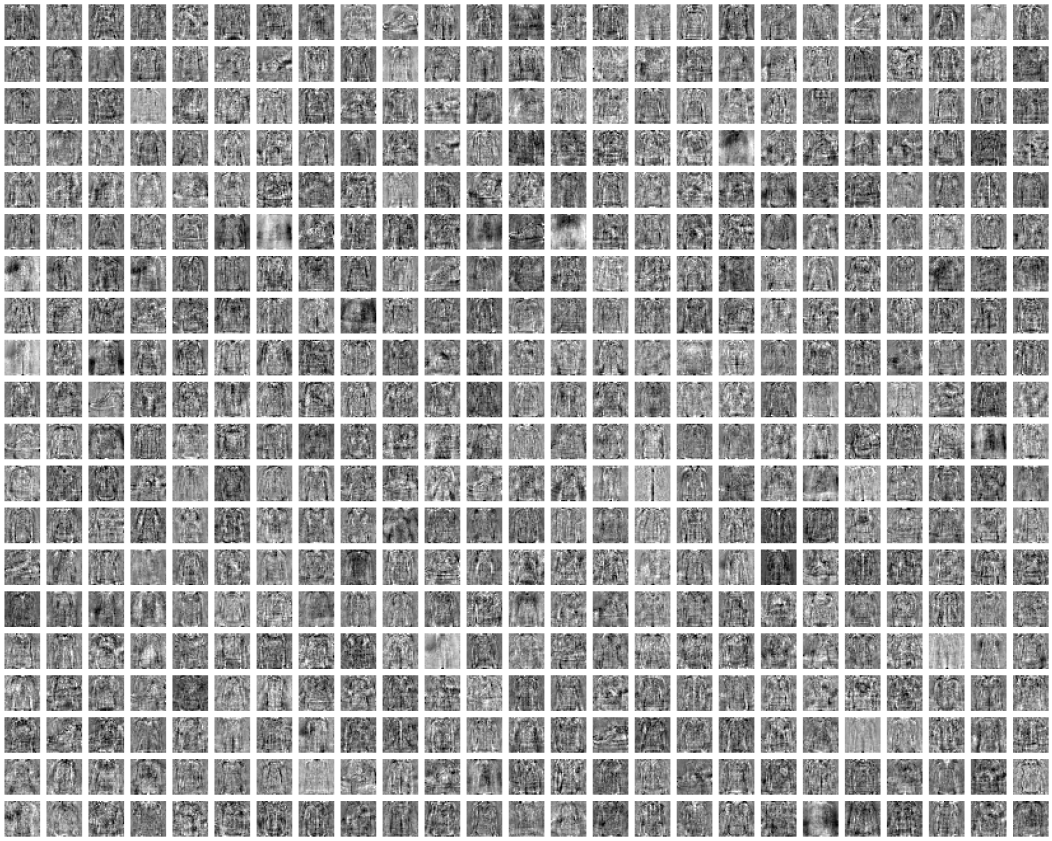


Figure 5: Visualization of the first hidden layer W_1. 500 features in total, each dimension 28×28.

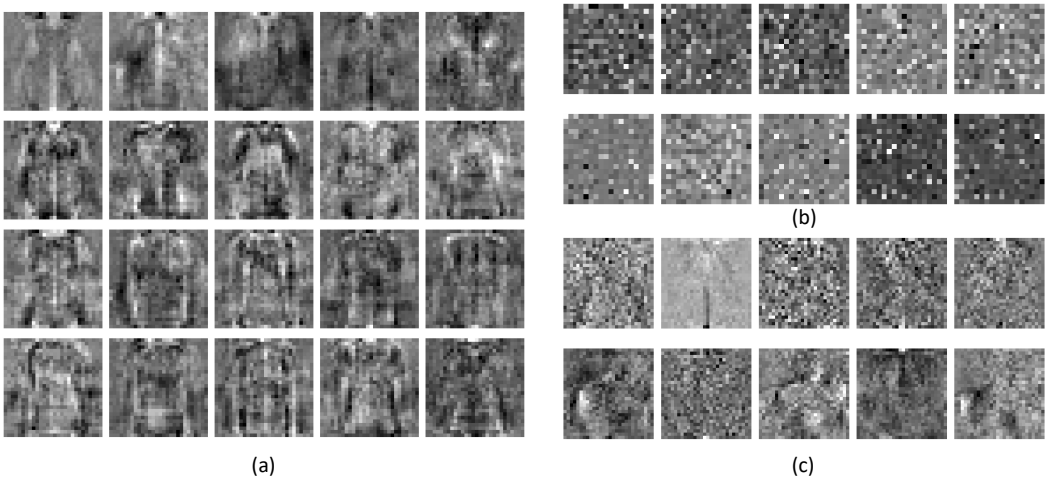


Figure 6: (a) Visualization of the compressed 20 features of first hidden layer W_1. (b) Visualization of the last hidden layer W_3. 10 patterns in total, each dimension 18×18. (c) Visualization of the combined transformation $W = W_1 \cdot W_2 \cdot W_3$, where 10 patterns, each dimension 28×28.

Acknowledgement

Thanks for the inspiring teaching of Professor Li Zhang.

References

[1] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.