Unveiling Derivatives in Deep and Convolutional Neural Networks: A Guide to Understanding and Optimization

Areeg Fahad Rasheed, M. Zarkoosh

Unveiling Derivatives in Deep and Convolutional Neural Networks: A Guide to Understanding and Optimization

Areeg Fahad Rasheed and M. Zarkoosh

[1] Al-Nahrain University, College of Information Engineering, Baghdad, Iraq

Email:areeg.fahad@coie-nahrain.edu.iq, fahedareeg@gmail.com

[2] Software Engineering, Baghdad, Iraq

Email:m94zarkoosh@gmail.com

Abstract

The world of artificial intelligence and deep learning is constantly evolving, with many pioneers and researchers utilizing frameworks such as TensorFlow and PyTorch to expedite their research. While these pre-built frameworks offer a swift execution process and alleviate the burden of pure programming, they may also result in a superficial grasp of the intricate mathematical mechanics involved in deep learning. This lack of deep understanding can impede the optimization process and hinder the achievement of optimal performance in developing deep networks. To overcome these challenges, the objective of this paper is to simplify, clarify, and remove obstacles to the mechanics of deep learning networks, streamlining the development process for researchers. In this paper, Our exploration of these networks will include explaining the derivative of various methods and activation functions, providing a deeper insight into the topic.

Keywords: *Deep learning, Convolutional neural network, PyTorch, TensorFlow*

1. Introduction

Deep learning is a subfield of machine learning that uses artificial neural networks to model complex patterns and relationships in data [1]. It has become increasingly popular in recent years due to its ability to handle large and diverse datasets and produce highly accurate predictions. Deep learning has been applied to a wide range of fields, including image and speech recognition [2], natural language processing, security [3–5], drug discovery, and even game playing [6]. It has also shown promise in healthcare, where it has been used for tasks such as medical image analysis and disease diagnosis [7]..etc.

Deep learning and neural networks rely on two important concepts: forward and backward propagation. During forward propagation, each layer of the network applies a set of mathematical operations to the input, transforming it into a new representation that is passed to the next layer. The output of the final layer is the prediction made by the neural network [8]. While in the backpropagation the error signal is propagated backwards through the network, allowing the network to adjust its weights in response to the error signal. This is done by computing the gradients of the loss function with respect to each weight in the network, and then adjusting the weights in the direction that reduces the loss function [9].

Building and developing a deep learning model using a pure programming language can be a time-consuming, complex, and error-prone task. This is where the use of deep learning frameworks such as TensorFlow, PyTorch, and many others can be immensely beneficial. These frameworks offer a variety of tools and resources that allow developers to build, test, and deploy deep learning models with ease. Many current researchers rely on these frameworks to accelerate their work and improve their produc-

tivity [10].

However, while deep learning frameworks offer many advantages, relying solely on them to build models can also be insufficient. It is important to have a solid understanding of the underlying principles of deep learning to optimise and fine-tune models. Additionally, there is a lack of research and literature available that explains the details of the forward and backward derivatives used in deep learning. Therefore, this paper aims to bridge this gap and provide a detailed explanation of the forward and backward derivative processes in deep learning, thus enabling researchers to gain a deeper understanding of how deep learning works and ultimately improving their ability to optimise and improve deep learning models.

In this paper, we will explain how the system produces the predicted value during the propagation process and adjusts the weights and bias during backpropagation. We will also discuss the use of multiple activation functions and network architectures. before diving into the details of deep learning, let's define the main concepts and notations.

2. Deep Learning Notation and Terminology

Before delving into the mathematical operations of deep learning, it is essential to understand the various notations and terminologies used in this field. In the following section, we provide brief definitions of these terminologies and their main purpose in deep learning technology.

2.1. The Deep Learning notation used in this paper

Table 1 presents the main notations used throughout this paper.

2.2. Deep learning terminologies

1) Weights: Weights, also known as parameters, are referred to as W: it is used in most of the deep learning and machine learning techniques. These weights determine the strength of the connections and play a critical role in the network's ability to accurately model the relationships between inputs and outputs. During the training process, the weights are updated based on the error between the network's predictions and the actual outputs. The goal of training is to adjust the weights to minimise this error and improve the accuracy of the model's predictions [11].
2) Bias b: It is a constant value that is added to the weighted sum of input before being fed to the activation function [12]. By allowing it to shift the output in the desired direction. It allows the model to be more flexible and accurate with the training data and prevents the over-fitting problem.
3) Activation function (a): is a mathematical function that is applied to the sum of weights of inputs and bias. Selecting the proper activation function is an important aspect of designing a deep learning model because it can have a significant impact on its performance. There are multiple activation functions used in deep learning models such as [13].
    - Sigmoid function: The sigmoid function takes an input and maps it to a value between 0 and 1. It is commonly used in the last layer for binary classification tasks [14].
    - ReLU (Rectified Linear Unit): ReLU takes the input and returns 0 if the input is negative or zero and the input value if the input is positive. It is popularly used in hidden layers because it is simple and computationally efficient [15].
    - Tanh (Hyperbolic tangent): Tanh is also used in the hidden layer more than the output layer. It takes the input and maps it to a value between -1 and 1 [16].
    - Softmax: Softmax takes the input and converts it to a vector between 0 and 1, where the total value of the vector is 1. It is usually used in the output layer and for multiple classification tasks [17].
4) Hidden Layers : The construction of a deep neural network consists of three types of layers. The first layer is the input layer, where the data is fed. The last layer is the output layer, where the network constructs the prediction. Between the two layers, there are multiple hidden layers. The hidden layers are called "hidden" because their output is not directly observed [18]. Instead, it is passed on to the next layer until it reaches the output layer. The purpose of the hidden layers is to transform the input data in a way that allows the network to

Table 1. Notation used in deep learning.

Symbol	Definition
A	Activation value in a neural network
C	Number of classes in a classification task
$J(y, \hat{y})$	Loss function
K	Convolution filter
L	The number of layers in a neural network
α	Learning rate
W	Weights of the model
X	Input to the model
Y_{pool}	Output of Pooling Layer
Z	Pre-activation values in a neural network
$(x^{(i)}, y^{(i)}) \in \mathcal{D}$	i-th data point in dataset
$\frac{\partial A}{\partial Z}$	the partial derivative of the Activation value A with respect to the Pre-activation value Z
$\frac{\partial J}{\partial A}$	the partial derivative of the loss function J with respect to the predicted output A
$\frac{\partial J}{\partial b}$	the partial derivative of the loss function J with respect to the bias b
$\frac{\partial J}{\partial W}$	the partial derivative of the loss function J with respect to the weight parameter W
$\frac{\partial J}{\partial Y_{pool}}$	the partial derivative of the loss function J with respect to the Output of Pooling Layer Y_{pool}
$\frac{\partial J}{\partial Z}$	the partial derivative of the loss function J with respect to the Pre-activation values Z

learn a complex representation of the data that can be used for classification, prediction, or other tasks. The number of hidden layers and the number of neurons in each hidden layer are hyperparameters that can be tuned to improve the neural network's performance.

5) Convolution layers: Convolutions layers are a type of layer commonly used in deep learning models for image recognition and computer vision tasks [19]. The main purpose of a convolution layer is to extract features from the input image by performing convolutional operations on it [20]. In a convolutions layer, the image is represented as a matrix of pixel values. The layer consists of a set of filters (kernels), which are small matrices that slide over the image and perform dot products with the corresponding input matrix. The dot product of the filter and a portion of the input matrix produces a scalar value, which is then used to create a new output matrix [21].
6) Gradient descent: It is an optimization algorithm used to minimize the cost function by adjusting the weights and biases of the neural network during the training process [22]. The algorithm works by calculating the gradient of the cost function concerning the weights and biases, which gives the direction of the steepest descent. The weights and biases are then updated in the opposite direction of the gradient, which moves them closer to the optimal values that minimize the cost function [23]. This process is repeated iteratively until the cost function is minimized.
7) Cost function: In deep learning, the method used to compute and evaluate the error between the predicted output and the actual output is known as the cost function. This function measures the discrepancy between the predicted output and the actual output and provides a measure of how well the neural network is performing [24]. The choice of cost function can depend on the specific problem being solved and the type of output produced by the network. For example, the mean squared error (MSE) cost function is commonly used in regression tasks, while the cross-entropy loss function is often used in classification tasks [25].
8) Forward propagation is the process of passing input data through a neural network to generate an output or prediction. During forward propagation, the input data is multiplied by the weights and added to the bias term at each layer [26]. Then, the activation function is applied to the weighted sum, producing the output for that layer. The output of one layer is then passed as input to the next layer until the output layer is reached and a final prediction is generated.
9) Back-propagation is a mathematical technique used to train deep neural networks. The purpose of back-propagation is to update the weights and biases of the neural network during training to minimize the error between the predicted output and the actual output. During backward propagation, the error between the predicted output and the actual output is computed, and then

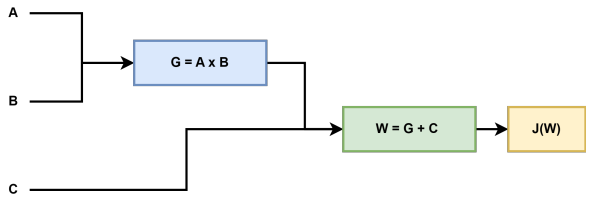

Figure 1. Chain rule example.

the error is propagated back through the network, starting from the output layer and working backward. As the error is propagated backward, the weights and biases of the network are updated using the gradient of the error concerning each weight and bias using the chain rule [27].

10) Regularization techniques play a crucial role in preventing overfitting, a common pitfall in machine learning that occurs when a model becomes overly tailored to the training data, compromising its ability to generalize to new, unseen examples. [1] To address this issue, regularization strategically introduces constraints or penalties during the learning process, guiding the model toward simpler and more generalizable solutions [28–30].

3. Chain rule principle

The chain rule is a calculus rule used to compute the derivative of a function composed of several nested functions [31]. In the context of neural networks, the chain rule is used to calculate the gradient of the cost function concerning the weights and biases of each layer in the network during backpropagation. It does this by multiplying the gradients of each layer together, starting from the output layer and working backward toward the input layer. To explain the concept of the chain rule, we will use a network diagram in Figure 1, consisting of multiple blocks, each representing a function. The output of one block will serve as the input to the next block, and so on.

The forward propagation process, as shown in figure 1, is used to compute the value of the final output, which in this case is $J(W)$. However, to update the parameters of the model during training, we need to compute the gradients of the loss function with respect to the parameters, starting from the output and working our way backward through the layers of the network.

In this case, the value of J depends on W, which is itself a function of G and C. Since C is a constant, it does not affect the calculation of the derivative of J with respect to W. Therefore, the chain rule can be used to express the derivative of J with respect to W as the product of the derivative of J with respect to G and the derivative of G with respect to W, as shown in equation (1):

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial G} \cdot \frac{\partial G}{\partial W} \tag{1}$$

To compute the derivative of J with respect to G, we follow a similar process. In the forward propagation, we see W depends on G and C. Therefore, we can compute the derivative of J with respect to G, as the product of the derivative of J with respect to W multiplied by the derivative of W, with respect to G, as shown in equation 2:

$$\frac{\partial J}{\partial G} = \frac{\partial J}{\partial W} \cdot \frac{\partial W}{\partial G} \tag{2}$$

Since we have already computed $\frac{\partial J}{\partial W}$ in equation 1, we now need to compute $\frac{\partial W}{\partial G}$. From equation 2, we know that W depends on G, so we can write: $W = G + C$ Taking the derivative of both sides with respect to G, we get: $\frac{\partial W}{\partial G} = 1$ Substituting this result back into equation 2, we get: $\frac{\partial J}{\partial G} = \frac{\partial J}{\partial W} \cdot 1 = \frac{\partial J}{\partial W}$ Therefore, we can conclude that: $\frac{\partial J}{\partial G} = \frac{\partial J}{\partial W}$

For the derivative of J with respect to C. Equation 3 is used.

$$\frac{\partial J}{\partial C} = \frac{\partial J}{\partial W} \cdot \frac{\partial W}{\partial C} \tag{3}$$

Since C is a constant, its derivative is zero. Therefore, the derivative of J with respect to C is 0.

To compute the derivatives of A and B with respect to J, we need to take into account the dependencies on W and G. We can calculate them using equations (4) and (5):

$$\frac{\partial J}{\partial A} = \frac{\partial J}{\partial G} \cdot \frac{\partial G}{\partial A} \tag{4}$$

$$\frac{\partial J}{\partial B} = \frac{\partial J}{\partial G} \cdot \frac{\partial G}{\partial B} \tag{5}$$

We can compute $\frac{\partial J}{\partial G}$ using equation (2), which is equal to $\frac{\partial J}{\partial W} \cdot \frac{\partial W}{\partial G}$. The $\frac{\partial G}{\partial A}$ is equal to B; therefore, the final equation of $\frac{\partial J}{\partial A}$ is $\frac{\partial J}{\partial A} = \frac{\partial J}{\partial W} \cdot \frac{\partial W}{\partial G} \cdot B$. Similarly, we can compute the $\frac{\partial J}{\partial B}$, which is equal to $\frac{\partial J}{\partial B} = \frac{\partial J}{\partial W} \cdot \frac{\partial W}{\partial G} \cdot A$.

4. Derivative Clarification

This section is divided into two parts. The first part provides an in-depth explanation of the derivatives in deep neural networks, covering multiple network architectures and different activation functions. The second part focuses on the derivatives of convolutional neural networks.

4.1. Deep Neural Networks

Deep neural networks have a wide range of architectures that can be used for various tasks. In this section, we will focus on explaining the most popular architectures used by researchers in the field.

1) Single-Layer Neural Network with Sigmoid Activation: It is the simplest neural network, using only one layer, the output layer, which uses a sigmoid activation function. In the forward process the input layer can consist of either a single value or a vector of values multiplied by weights and added to the bias term to generate the output linear value Z as shown in equation (7). The resulting sum is then passed through a sigmoid activation function, which generates the predicted output value, denoted as $\hat{y}$ or A as in equation (8).

$$Z = W^T \cdot X + b \tag{7}$$

$$A \; or \; \hat{Y} = \frac{1}{1 + e^{-Z}} \tag{8}$$

In deep learning, the ultimate goal is to train a model that can accurately predict the desired output for a given input. To achieve this, we need to update the model's weights based on the difference between the predicted output and the actual output. The cost function, also known as the loss function, is used to calculate this difference. The log loss function also commonly referred to as binary cross-entropy loss is used for this architecture as in equation (9).

$$J(A, Y) = Y log(A) + (1 - Y) log(1 - A) \tag{9}$$

In order to update the weights W in the neural network during backpropagation, we need to compute the gradient of the loss function (J) with respect to the weights W. In order to do this, we need to compute the derivatives of J with respect to A, then the derivative of Z with respect to A, and finally the derivative of J with respect to W using the chain rule.
Based on equation (9) and using calculus, we can compute the derivative of $\frac{\partial J}{\partial A}$ as shown in equation (10)

$$\frac{\partial J}{\partial A} = -\frac{Y}{A} + \frac{1 - Y}{1 - A} \tag{10}$$

After computing $\frac{\partial J}{\partial A}$, we need to compute $\frac{\partial J}{\partial Z}$ according to equation (11).

$$\frac{\partial J}{\partial Z} = \frac{\partial J}{\partial A} \cdot \frac{\partial A}{\partial Z} \tag{11}$$

the $\frac{\partial J}{\partial A}$ is equal to $(-\frac{Y}{A} + \frac{1-Y}{1-A})$ while $\frac{\partial A}{\partial Z}$ is the derivative of sigmoid function in equation (8) which is equal to $A(1 - A)$ as illustrated in equation (12)

$$\begin{aligned}
\frac{\partial A}{\partial Z}\sigma(Z) &= \frac{\partial A}{\partial Z}\left[\frac{1}{1 + e^{-Z}}\right] \\
&= \frac{\partial A}{\partial Z}\left(1 + \mathrm{e}^{-Z}\right)^{-1} \\
&= -(1 + e^{-Z})^{-2}(-e^{-Z}) \\
&= \frac{e^{-Z}}{(1 + e^{-Z})^2} \\
&= \frac{1}{1 + e^{-Z}} \cdot \frac{e^{-Z}}{1 + e^{-Z}} \\
&= \frac{1}{1 + e^{-Z}} \cdot \frac{(1 + e^{-Z}) - 1}{1 + e^{-Z}} \\
&= \frac{1}{1 + e^{-Z}} \cdot \left(\frac{1 + e^{-Z}}{1 + e^{-Z}} - \frac{1}{1 + e^{-Z}}\right) \\
&= \frac{1}{1 + e^{-Z}} \cdot \left(1 - \frac{1}{1 + e^{-Z}}\right) \\
&= \sigma(Z) \cdot (1 - \sigma(Z)) \\
&= A(1 - A)
\end{aligned} \tag{12}$$

Then the $\frac{\partial J}{\partial Z}$ is equal to $-\frac{Y}{A} + \frac{1-Y}{1-A} \cdot A(1 - A)$
Once we have computed the derivative of the loss function with respect to the output of the linear transformation (i.e., $\frac{\partial J}{\partial Z}$), we can use the chain rule of differentiation to compute the derivatives of the loss function with respect to the weight $\frac{\partial J}{\partial W}$ and biases $\frac{\partial J}{\partial b}$ of the neural network, according to equations (13) and (14) respectively.

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Z} \cdot \frac{\partial Z}{\partial W} \tag{13}$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial Z} \cdot \frac{\partial Z}{\partial b} \tag{14}$$

the value of $\frac{\partial Z}{\partial W}$ is equal to $\frac{\partial Z}{\partial W}(W^T \cdot X + b) = X$ and the $\frac{\partial Z}{\partial b}$ is equal to $\frac{\partial Z}{\partial b}(W^T \cdot X + b) = 1$ therefore the final value of the $\frac{\partial Z}{\partial W}$ and $\frac{\partial Z}{\partial b}$ are

$$\frac{\partial J}{\partial W} = -\frac{Y}{A} + \frac{1-Y}{1-A}A(1-A)X$$
$$\frac{\partial J}{\partial b} = -\frac{Y}{A} + \frac{1-Y}{1-A}A(1-A)$$

After computing the gradients $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$, we can adjust the weights and biases using equations (15) and (16) respectively. These updates aim to decrease the cost error and improve the predictive performance of the system.

$$W = W - \alpha\frac{\partial J}{\partial W} \tag{15}$$

$$b = b - \alpha\frac{\partial J}{\partial b} \tag{16}$$

If $\frac{\partial J}{\partial W}$ is greater than zero, it means that increasing the value of the weights W would increase the value of the cost function J. Therefore, the weights need to be decreased to minimize the cost.
On the other hand, if $\frac{\partial J}{\partial W}$ is less than zero, it means that decreasing the value of the weights W would increase the value of the cost function J. Therefore, the weights need to be increased to minimize the cost.

2) Two-Layer Neural Network with Tanh and Sigmoid Activations: The architecture of this neural network differs from the first one as it consists of two layers and utilizes two activation functions. The first is the hyperbolic tangent (Tanh) function (refer to equation 17), while the second is the sigmoid function (refer to equation 8), used in the output layer. The forward equation begins by computing Z and then applies the activation function to calculate the value (A) of the first layer A. This activation output A then serves as the input for the second layer. The loss function in this architecture is the same as the one in equation (9). Table 2 illustrates the forward and backward equations of this architecture.

$$A^{[l]}_{Tanh} = \frac{e^{Z^{[l]}} - e^{-Z^{[l]}}}{e^{Z^{[l]}} + e^{-Z^{[l]}}} \tag{17}$$

It is important to know that the derivative of activation function A with respect to Z based on equation (17) is computed as in equation (18):

$$\begin{aligned}\frac{\partial A}{\partial Z} &= \frac{(e^Z + e^{-Z})(e^Z + e^{-Z}) - (e^Z - e^{-Z})(e^Z - e^{-Z})}{(e^Z + e^{-Z})^2} \\ &= \frac{(e^{2Z} + e^{-2Z}) - (e^{2Z} - e^{-2Z})}{(e^{2Z} + e^{-2Z})} \\ &= \frac{(e^{2Z} + e^{-2Z})}{(e^{2Z} + e^{-2Z})} - \frac{(e^{2Z} - e^{-2Z})}{(e^{2Z} + e^{-2Z})} \\ &= 1 - \tanh(Z)^2 = 1 - A^2 \end{aligned} \tag{18}$$

3) A Two-Layer Neural Network with ReLU and Sigmoid Activations involves a process of computing both forward and backward propagation, similar to the previous architecture, as summarized in Table 3. The first layer employs the ReLU activation function, and the second layer uses the sigmoid activation function. However, the key distinction lies in the equations and derivatives of the ReLU activation function with respect to Z, which are presented in equations (19) and (20), respectively.

$$A = \begin{cases} Z, & \text{if } Z > 0 \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

$$\frac{\partial A}{\partial Z} = \begin{cases} 1, & \text{if } Z > 0 \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

4) A Multi-Layer Neural Network with ReLU and Sigmoid Activations. It is more appropriate for the complex tasks. In this architecture, the ReLU activation function is applied to the hidden layers of the network, while the sigmoid function is used for the output layer. The equations and derivatives for both activation functions can be summarized in Table 4.

5) Multi-Layer Neural Network with ReLU and Softmax Activations. This architecture is used for multiclass classification tasks. In this architecture, the softmax equation in (20) produces a vector, and the predicted class is determined by selecting the index of the highest value in the vector. The loss

Table 2. Forward and Backward Equations for a Two-Layer Perceptron with Tanh and Sigmoid Activation Functions

Layer Name	Forward Equation	Backward Equation
Layer 1	$W^{[1]}_{current}$	$W^{[1]}_{new} := W^{[1]}_{current} - \alpha \frac{\partial J}{\partial W^{[1]}_{current}}$
		$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot X^T$
	$b^{[1]}_{current}$	$b^{[1]}_{new} := b^{[1]}_{current} - \alpha \frac{\partial J}{\partial b^{[1]}_{current}}$
		$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot 1$
	$Z^{[1]} = XW^{[1]} + b^{[1]}$	$\frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot (1 - (A^{[1]})^2)$
	$A^{[1]} = \frac{e^{Z^{[1]}} - e^{-Z^{[1]}}}{e^{Z^{[1]}} + e^{-Z^{[1]}}}$	$\frac{\partial J}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot W^{[2]}$
Layer 2	$W^{[2]}_{current}$	$W^{[2]}_{new} := W^{[2]}_{current} - \alpha \frac{\partial J}{\partial W^{[2]}_{current}}$
		$\frac{\partial J}{\partial W^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot A^{[2]}$
	$b^{[2]}_{current}$	$b^{[2]}_{new} := b^{[2]}_{current} - \alpha \frac{\partial J}{\partial b^{[2]}_{current}}$
		$\frac{\partial J}{\partial b^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot 1$
	$Z^{[2]} = A^{[1]}W^{[2]} + b^{[2]}$	$\frac{\partial J}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot A^{[2]}(1 - A^{[2]})$
	$A^{[2]} = \frac{1}{1+e^{-Z^{[2]}}}$	$\frac{\partial J}{\partial A^{[2]}} = -\frac{Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}}$
Loss Function	$J(A^{[2]}, Y) = Y \log(A^{[2]}) + (1-Y) \log(1 - A^{[2]})$	

Table 3. Forward and Backward Equations for a Two-Layer Perceptron with RelU and Sigmoid Activation Functions

Layer Name	Forward Equation	Backward Equation
Layer 1	$W^{[1]}_{current}$	$W^{[1]}_{new} := W^{[1]}_{current} - \alpha \frac{\partial J}{\partial W^{[1]}_{current}}$
		$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot X^T$
	$b^{[1]}_{current}$	$b^{[1]}_{new} := b^{[1]}_{current} - \alpha \frac{\partial J}{\partial b^{[1]}_{current}}$
		$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot 1$
	$Z^{[1]} = XW^{[1]} + b^{[1]}$	$\frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[1]} = \begin{cases} Z^{[1]}, & \text{if } Z^{[1]} \geq 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot W^{[2]}$
Layer 2	$W^{[2]}_{current}$	$W^{[2]}_{new} := W^{[2]}_{current} - \alpha \frac{\partial J}{\partial W^{[2]}_{current}}$
		$\frac{\partial J}{\partial W^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot A^{[2]}$
	$b^{[2]}_{current}$	$b^{[2]}_{new} := b^{[2]}_{current} - \alpha \frac{\partial J}{\partial b^{[2]}_{current}}$
		$\frac{\partial J}{\partial b^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}_{current}} = \frac{\partial J}{\partial Z^{[2]}} \cdot 1$
	$Z^{[2]} = A^{[1]}W^{[2]} + b^{[2]}$	$\frac{\partial J}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot A^{[2]}(1 - A^{[2]})$
	$A^{[2]} = \frac{1}{1+e^{-Z^{[2]}}}$	$\frac{\partial J}{\partial A^{[2]}} = -\frac{Y}{A^{[2]}} + \frac{1-Y}{1-A^{[2]}}$
Loss Function	$J(A^{[2]}, Y) = Y \log(A^{[2]}) + (1-Y) \log(1 - A^{[2]})$	

function for softmax is typically the cross-entropy loss, which is given by equation (21).

$$A = \begin{bmatrix} \frac{e^{Z_1}}{\sum_i^C e^{z_i}} \\ \frac{e^{Z_2}}{\sum_i^C e^{z_i}} \\ \vdots \\ \frac{e^{Z_C}}{\sum_i^C e^{z_i}} \end{bmatrix} \tag{20}$$

$$L(Y, A) = -\sum_{i=1}^{C} Y_i \log(A_i) \tag{21}$$

In backpropagation, we need to compute the gradients of the loss function with respect to all the parameters of the model, including A and Z. The gradient of the loss function with respect to A can be computed using equation 22, which represents the cross-entropy loss for a multi-class classification

problem. The second important step is to compute the derivative of A with respect to Z. The equation for the softmax function (A) yields a vector with multiple elements, where each element corresponds to the predicted probability of belonging to a specific class as in equation (20). The predicted probability is obtained by taking the exponential of the Z value for each class and dividing it by the sum of the exponentials of all Z values. Thus, to calculate the derivative of a specific class A_i, it is necessary to sum the derivative of A_i with respect to all Z. This is because every element of A depends on all elements of Z, and a change in any Z value will affect all the elements of A. The Jacobian matrix in equation (23) represents all the derivatives that need to be computed to obtain the general derivative of $\frac{\partial A}{\partial Z}$, which is important in back propagation algorithms to update the model's parameters during training.

$$\frac{\partial L}{\partial A} = -\sum_{i=1}^{C} y_i \log(a_i) \tag{22}$$

$$\frac{\partial A}{\partial Z} = \begin{bmatrix} \frac{\partial a_1}{\partial z_1} & \frac{\partial a_1}{\partial z_2} & \cdots & \frac{\partial a_1}{\partial z_c} \\ \frac{\partial a_2}{\partial z_1} & \frac{\partial a_2}{\partial z_2} & \cdots & \frac{\partial a_2}{\partial z_c} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial a_c}{\partial z_1} & \frac{\partial a_c}{\partial z_2} & \cdots & \frac{\partial a_c}{\partial z_c} \end{bmatrix} \tag{23}$$

Each element in the Jacobian matrix, $\frac{\partial A}{\partial Z}$, can be simplified. To illustrate this, we can take the values of $\frac{\partial a_1}{\partial z_1}$, $\frac{\partial a_1}{\partial z_2}$, $\frac{\partial a_2}{\partial z_1}$, and $\frac{\partial a_2}{\partial z_2}$ and simplify them as in equation (24, 25, 26, 27)

$$\begin{aligned} \frac{\partial a_1}{\partial z_1} &= \frac{\partial}{\partial z_1}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}}\right) \\ &= \frac{e^{z_1}\cdot\frac{\partial}{\partial z_1}(e^{z_1})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= \frac{e^{z_1}(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}} \cdot \frac{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}-e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}} \\ &= a_1(1-a_1) \end{aligned} \tag{24}$$

¡

$$\begin{aligned} \frac{\partial a_1}{\partial z_2} &= \frac{\partial}{\partial z_2}\left(\frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}}\right) \\ &= \frac{e^{z_1}\cdot\frac{\partial}{\partial z_2}(e^{z_2})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= \frac{-e^{z_1}e^{z_2}}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= -a_1a_2 \end{aligned} \tag{25}$$

¡

$$\begin{aligned} \frac{\partial a_2}{\partial z_1} &= \frac{\partial}{\partial z_1}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}}\right) \\ &= \frac{e^{z_2}\cdot\frac{\partial}{\partial z_1}(e^{z_1})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= \frac{-e^{z_1}e^{z_2}}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\ &= -a_1a_2 \end{aligned} \tag{26}$$

$$
\begin{aligned}
\frac{\partial a_2}{\partial z_2} &= \frac{\partial}{\partial z_2}\left(\frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}}\right) \\
&= \frac{e^{z_2}\cdot\frac{\partial}{\partial z_2}(e^{z_2})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\
&= \frac{e^{z_2}(e^{z_2}+e^{z_1}+e^{z_3}+...+e^{z_n})}{(e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n})^2} \\
&= \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}}\cdot\frac{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}-e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}+...+e^{z_n}} \\
&= a_2(1-a_2)
\end{aligned} \tag{27}
$$

¡

$$
\frac{\partial a_i}{\partial z_j} = \begin{cases} -a_i a_j & \text{if } i \neq j \\ a_i(1-a_j) & \text{if } i = j \end{cases} \tag{28}
$$

¡

$$
\frac{\partial A}{\partial Z} = \begin{bmatrix} \frac{\partial a_1}{\partial z_1} & \frac{\partial a_1}{\partial z_2} & \cdots & \frac{\partial a_1}{\partial z_n} \\ \frac{\partial a_2}{\partial z_1} & \frac{\partial a_2}{\partial z_2} & \cdots & \frac{\partial a_2}{\partial z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial a_c}{\partial z_1} & \frac{\partial a_c}{\partial z_2} & \cdots & \frac{\partial a_c}{\partial z_c} \end{bmatrix} = \begin{bmatrix} a_1(1-a_1) & -a_1a_2 & \cdots & -a_1a_c \\ -a_1a_2 & a_2(1-a_2) & \cdots & -a_2a_c \\ \vdots & \vdots & \ddots & \vdots \\ -a_ca_1 & -a_ca_2 & \cdots & a_c(1-a_c) \end{bmatrix} \tag{28}
$$

¡

$$
\begin{aligned}
&\frac{\partial J}{\partial z_j} = \frac{\partial J}{\partial A}\cdot\frac{\partial A}{\partial z_j} \\
&\quad \frac{\partial J}{\partial A} = -\sum_{i=1}^{C}\frac{y_i}{a_i} \rightarrow \frac{\partial A}{\partial z_j} = \sum_{i=1}^{C}\frac{\partial a_i}{\partial z_j} \\
&\quad Therefore \\
&\quad = (-\sum_{i=1}^{C}\frac{y_i}{a_i})\cdot\sum_{i=1}^{C}\frac{\partial a_i}{\partial z_j} \\
&\quad Simplification \\
&\quad = \sum_{i=1}^{C}\frac{y_i}{a_i}a_i a_j i \neq j - \frac{y_i}{a_i}a_i(1-a_i) \\
&\quad = \sum_{i=1}^{C} y_i a_j + y_i a_i - y_i \rightarrow = a_j\sum_{i=1}^{C} y_i + \sum_{i=1}^{C} y_i a_i - \sum_{i=1}^{C} y_i \\
&\quad = a_j\sum_{i=1}^{C} y_i + \sum_{i=1}^{C} y_i a_i - 1 \\
&\quad = a_j - y_j
\end{aligned} \tag{22}
$$

Then, from equation (29), we can produce the general equation for $\frac{\partial J}{\partial Z}$ as shown in equation (30).

$$\frac{\partial J}{\partial Z} = A - Y \tag{30}$$

After computing the derivative of $\frac{\partial J}{\partial Z}$, the derivatives of $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$ of softmax can be computed similar to the previous architectures as shown in equations (31) and (32). These derivatives can then be used to update the weights and biases as in equation (33), (34) respectively.

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Z} \cdot \frac{\partial Z}{\partial W} = (A - Y) \cdot A^{[L-1]} \tag{31}$$

$$\frac{\partial J}{\partial b^{[L]}} = \frac{\partial J}{\partial Z} \cdot \frac{\partial Z}{\partial b^{[L]}} = (A - Y) \cdot 1 = A - Y \tag{32}$$

$$W^{[L]}_{new} = W^{[L]}_{current} - \alpha \frac{\partial J}{\partial W^{[L]}} \tag{33}$$

$$b^{[L]}_{new} = b^{[L]}_{current} - \alpha \frac{\partial J}{\partial b^{[L]}} \tag{34}$$

Table 5 summarizes the forward and backward equations for multilayer ReLU and softmax; the equations of layer 1 until layer L-1 is the same as in Table 3 since both use ReLU as activation function, while the equations for layer L is different because softmax is used instead of a sigmoid.

4.2. Convolution Neural Networks

A Convolutional Neural Network (CNN) is a type of deep neural network that is specifically designed for image processing and recognition tasks. CNNs differ from traditional deep neural networks in that they contain specialized layers that are specifically designed for image processing. In a typical CNN, there are two main types of layers:

1) Convolutional Layers: These layers perform convolution operations between the input image and a set of learnable filters (also called kernels or weights) that slide over the input image to extract features. The output of each filter is a feature map that represents a specific aspect of the input image. The convolution operation preserves the spatial relationship between pixels in the input image and helps to extract local features.
2) Pooling Layers: These layers are typically used after the convolutional layers to reduce the size of the feature maps and to introduce some form of translational invariance. The most common type of pooling layer is the MaxPooling layer, which selects the maximum value from a local neighborhood of the feature map. This operation helps to reduce the spatial dimensionality of the feature map and makes the network more robust to small translations in the input image.

In addition to these two types of layers, CNNs can also include fully connected layers, similar to those found in traditional deep neural networks. These layers perform a linear transformation on the output of the preceding layer and apply a non-linear activation function (such as the sigmoid or ReLU function) to introduce non-linearity into the model. The output of the final fully connected layer is then fed into an activation function, which produces a probability distribution over the possible classes of the input image.

For the derivative clarification, two different network architectures will be used in this paper. The first architecture consists of the following layers: convolutional layer, max pooling layer, fully connected layer with ReLU activation function, and sigmoid output layer. The second architecture is similar to the first one, but it uses softmax activation function in the last layer instead of sigmoid activation.

1) CNN with max pooling, ReLU and sigmoid: During the forward propagation the first step is to convolve a set of learnable filters (also known as kernels (K)) over the input sample (X) to generate the output feature maps (Y). This is the core operation of the convolutional layer in a CNN, and it is used to extract local features from the input image. During the convolution operation, the kernel slides over the input sample, multiplying the values of the kernel with the corresponding values in the input sample and summing the results to produce a single output value as shown in the equation (35) . This process is repeated for each position in the input sample, resulting in a feature map that highlights certain features or patterns in the input.

$$z_{1[i,j]} = B + \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x_{(i+a,j+b)} * K_{(a,b)} \tag{35}$$

In this equation, the B is the bias, k is the size of the kernel, $x_{i,j}$ is the value of the input feature map at position (i, j), $K_{(a,b)}$ is the value of the kernel at position (a, b), and $Z_{1[i,j]}$ is the value of the output feature map at position (i, j). The ReLU activation function is applied to the output feature map Y after the convolution equation (36), and then the resulting feature map is passed to the max pooling layer. to reduce the spatial dimensionality of the feature map as in equation (37).

$$A_{1[i,j]} = Max(0, z_{1[i,j]}) \tag{36}$$

$$y_{pool}(i, j) = \max_{a=0}^{p-1} \max_{b=0}^{p-1} y_{relu}(i \times s + a, j \times s + b) \tag{37}$$

where Y is the input feature map, Y_{pool} is the output feature map after pooling, p is the pooling size (e.g., 2×2 pooling), s is the stride length, and (i, j) are the indices of the output feature map. After applying the max pooling layer, the resulting feature map is typically flattened into a one-dimensional vector, which serves as the input to the fully connected layers. The flattening operation reshapes the 2D or 3D tensor output of the max pooling layer into a 1D vector. Then the output vector will be fed to the deep neural network, following the same process as in the previous sections (i.e, the output of the max pooling layer y_pool is flattened into a one-dimensional vector, denoted as A, which is multiplied by the weights and added to the bias term, and then fed into the activation function). as in equation (38, 39, 40, 40)

$$Y_{pool} = \text{maxpool}(Y_{relu}) \tag{38}$$

$$Z_2 = Y_{pool} \cdot W + b_2 \tag{39}$$

$$A_2 = \text{sigmoid}(Z_2) \tag{40}$$

$$J(A_2, Y) = -Y \log(A_2) - (1 - Y) \log(1 - A_2) \tag{41}$$

During the back propagation, the derivatives of the loss function and the fully connected layer are the same as in a deep neural network. The difference lies in the process of computing the derivatives of the max pooling and convolution layers. The first derivative computed during backpropagation is the loss function with respect to the predicted output $\frac{\partial J}{\partial A_2}$ as shown in equation (42)

$$\frac{\partial J}{\partial A_2} = -\frac{Y}{A_2} + \frac{1 - Y}{1 - A_2} \tag{42}$$

Next, we need to compute the derivatives of $\frac{\partial A_2}{\partial Z_2}$, $\frac{\partial J}{\partial Z_2}$, $\frac{\partial J}{\partial Z_2}$, $\frac{\partial J}{\partial W}$, and $\frac{\partial J}{\partial b_2}$ in order to complete the back propagation process for the fully connected network. as shown in equations (43, 44, 45, 46, 47)

$$\frac{\partial A_2}{\partial Z_2} = A_2(1 - A_2) \tag{43}$$

$$\frac{\partial A_2}{\partial Z_2} = A_2(1 - A_2) \tag{44}$$

$$\frac{\partial J}{\partial Z_2} = \frac{\partial J}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} = \frac{\partial J}{\partial A_2} \cdot A_2(1 - A_2) \tag{45}$$

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W} = \frac{\partial J}{\partial Z_2} \cdot Y_{pool} \tag{46}$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial b_2} = \frac{\partial J}{\partial Z_2} \cdot 1 \tag{47}$$

To compute the derivatives of the convolution and max pooling layers, we begin by computing $\frac{\partial J}{\partial Y_{pool}}$ using equation (48), which involves multiplying $\frac{\partial J}{\partial Z_2}$ by $\frac{\partial Z_2}{\partial Y_{pool}}$

$$\frac{\partial J}{\partial Y_{pool}} = \frac{\partial J}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial Y_{pool}} = \frac{\partial J}{\partial Z_2} \cdot W \tag{48}$$

To compute the derivative of J with respect to Y_{relu}, we need to evaluate the product of $\frac{\partial J}{\partial Y_{pool}}$ and $\frac{\partial Y_{pool}}{\partial Y_{relu}}$ as shown in equation (49). The value of $\frac{\partial J}{\partial Y_{pool}}$ is computed in the previous step (eqaution 48), while $\frac{\partial Y_{pool}}{\partial Y_{relu}}$ is a matrix that has the same shape as Y_{relu} and each element of the matrix is either 0 or 1. The value 1 appears at the location of the maximum value within each pooling region, and 0s appear elsewhere see the equation (50).

$$\frac{\partial J}{\partial Y_{relu}} = \frac{\partial J}{\partial Y_{pool}} \cdot \frac{\partial Y_{pool}}{\partial Y_{relu}} = \frac{\partial J}{\partial Y_{pool}} \cdot M \tag{49}$$

$$\frac{\partial Y_{pool}}{\partial Y_{relu}} = M_{i,j,k} = \begin{cases} 1 & \text{if } Y_{relu}(i, j, k) = \max_{p,q} Y_{relu}(p, q, k) \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

the indices (i, j, k) refer to the ith row, jth column, and kth channel of the tensors. The symbol $\max_{p,q}$ denotes the maximum operation over the pooling region that corresponds to the output location (i, j, k) in Y_{pool}.
The $\frac{\partial J}{\partial Z_1}$ is computed using equation (51)

$$\frac{\partial J}{\partial Z_1} = \frac{\partial J}{\partial Y_{relu}} \cdot \frac{\partial Y_{relu}}{\partial Z_1} = \frac{\partial J}{\partial Y_{relu}} \cdot \begin{cases} 1 & \text{if } z1_{i,j,k} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{51}$$

After computing $\frac{\partial J}{\partial Z_1}$, it is time to complete the derivative of the remaining parameters, which are $\frac{\partial J}{\partial K}$, $\frac{\partial J}{\partial b_1}$, and $\frac{\partial J}{\partial X}$. The first one is the simplest, which is $\frac{\partial J}{\partial b_1}$. The bias value is usually a single value that is broadcast across all input pixels and used for all depths. From the forward propagation equation (35), we can notice that the bias affects only Z_1 and has no effect on the input X or kernel K. Since it is only one value, the equation for computing the bias is shown in equation (52).

$$\begin{aligned} \frac{\partial J}{\partial b_1} &= \frac{\partial J}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial b_1} \\ &= \frac{\partial J}{\partial Z_1} \cdot 1 \\ &= \sum_{i=1}^{n_H} \sum_{j=1}^{n_W} \frac{\partial J}{\partial Z_1[i,j]} \end{aligned} \tag{52}$$

The second derivative required is the $\frac{\partial J}{\partial K}$ let simplify the equation 35 to make the general equation

$$z_{1[i,j]} = b_1 + \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x_{(i+a,j+b)} * K_{(a,b)} \tag{35}$$

$$\begin{aligned} Z_{1[0,0]} &= x_{[0,0]}k_{[0,0]} + x_{[0,1]}k_{[0,1]} + \cdots + x_{[0+k-1,0+k-1]}k_{[k-1,k-1]} + b_1 \\ Z_{1[0,1]} &= x_{[0,1]}k_{[0,0]} + x_{[0,2]}k_{[0,1]} + \cdots + x_{[0+k-1,0+k-1]}k_{[k-1,k-1]} + b_1 \\ Z_{1[1,0]} &= x_{[1,0]}k_{[0,0]} + x_{[1,1]}k_{[0,1]} + \cdots + x_{[1+K-1,2+k-1]}k_{[k-1,k-1]} + b_1 \\ Z_{1[1,1]} &= x_{[1,1]}k_{[0,0]} + x_{[1,2]}k_{[0,1]} + \cdots + x_{[2+k-1,2+k-1]}k_{[k-1,k-1]} + b_1 \\ &\vdots \\ Z_{1[i,j]} &= x_{[i+0,j+0]}k_{[0,0]} + x_{[i+0,1+j]}k_{[0,1]} + \cdots + x_{[i+k-1,j+k-1]}k_{[k-1,k-1]} + b_1 \end{aligned}$$

¡

$$\frac{\partial J}{\partial k_{[0,0]}} = \frac{\partial J}{\partial z_{1[0,0]}} \cdot \frac{\partial z_{1[0,0]}}{\partial x_{[0,0]}} + \frac{\partial J}{\partial z_{1[0,1]}} \cdot \frac{\partial z_{1[0,1]}}{\partial x_{[0,1]}} + \frac{\partial J}{\partial z_{1[1,0]}} \cdot \frac{\partial z_{1[1,0]}}{\partial x_{[1,0]}} + ... + \frac{\partial J}{\partial z_{1[n_{H-1},n_{W-1}]}} \cdot \frac{\partial z_{1[n_{H-1},n_{W-1}]}}{\partial x_{[0+n_{H-1},0+n_{W-1}]}} \tag{53}$$

¡

$$\frac{\partial J}{\partial k_{[0,1]}} = \frac{\partial J}{\partial z_{1[0,0]}} \cdot \frac{\partial z_{1[0,0]}}{\partial x_{[0,1]}} + \frac{\partial J}{\partial z_{1[0,1]}} \cdot \frac{\partial z_{1[0,1]}}{\partial x_{[0,2]}} + \frac{\partial J}{\partial z_{1[1,0]}} \cdot \frac{\partial z_{1[1,0]}}{\partial x_{[1,1]}} + ... + \frac{\partial J}{\partial z_{1[n_{H-1},n_{W-1}]}} \cdot \frac{\partial z_{1[n_{H-1},n_{W-1}]}}{\partial x_{[0+n_{H-1},1+n_{W-1}]}} \tag{54}$$

Then, the general form to compute the derivative of K with respect to the loss

$$
\begin{aligned}
\frac{\partial J}{\partial K} &= \frac{\partial J}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial K} && \text{where } \frac{\partial Z_1}{\partial K} = X \\
\frac{\partial J}{\partial K} &= \frac{\partial J}{\partial Z_1} \cdot X \\
&= \\
\frac{\partial J}{\partial k_{[p,q]}} &= \sum_{i=0}^{n_H-1} \sum_{j=0}^{n_W-1} \frac{\partial J}{\partial z_{1[i,j]}} \cdot x_{[i+p,j+q]} && (55)
\end{aligned}
$$

¡

$$
\begin{aligned}
Z_{1[0,0]} &= x_{[0,0]}k_{[0,0]} + x_{[0,1]}k_{[0,1]} + \cdots + x_{[0+k-1,0+k-1]}k_{[k-1,k-1]} + b_1 \\
Z_{1[0,1]} &= x_{[0,1]}k_{[0,0]} + x_{[0,2]}k_{[0,1]} + \cdots + x_{[0+k-1,0+k-1]}k_{[k-1,k-1]} + b_1 \\
Z_{1[1,0]} &= x_{[1,0]}k_{[0,0]} + x_{[1,1]}k_{[0,1]} + \cdots + x_{[1+K-1,2+k-1]}k_{[k-1,k-1]} + b_1 \\
Z_{1[1,1]} &= x_{[1,1]}k_{[0,0]} + x_{[1,2]}k_{[0,1]} + \cdots + x_{[2+k-1,2+k-1]}k_{[k-1,k-1]} + b_1 \\
&\vdots \\
Z_{1[i,j]} &= x_{[i+0,j+0]}k_{[0,0]} + x_{[i+0,1+j]}k_{[0,1]} + \cdots + x_{[i+k-1,j+k-1]}k_{[k-1,k-1]} + b_1
\end{aligned}
$$

¡

$$
\frac{\partial J}{\partial x_{[0,0]}} = \frac{\partial J}{\partial z_{1[0,0]}} \cdot \frac{\partial z_{1[0,0]}}{\partial x_{[0,0]}} = \frac{\partial J}{\partial z_{1[0,0]}} \cdot k_{[0,0]} \tag{58}
$$

¡

$$
\frac{\partial J}{\partial x_{[0,1]}} = \frac{\partial J}{z_{1[0,0]}} \cdot \frac{\partial z_{1[0,0]}}{x_{[0,1]}} + \frac{\partial J}{z_{1[0,1]}} \cdot \frac{\partial z_{1[0,1]}}{x_{[0,1]}} = \frac{\partial J}{z_{1[0,0]}} \cdot k_{[0,1]} + \frac{\partial J}{z_{1[0,1]}} \cdot k_{[0,0]} \tag{59}
$$

¡

$$
\frac{\partial J}{\partial x_{[1,0]}} = \frac{\partial J}{\partial z_{1[1,0]}} \cdot \frac{\partial z_{1[1,0]}}{\partial x_{[1,0]}} = \frac{\partial J}{\partial z_{1[1,0]}} \cdot k_{[0,0]} \tag{60}
$$

¡

$$
\frac{\partial J}{\partial x_{[1,1]}} = \frac{\partial J}{\partial z_{1[1,0]}} \cdot \frac{\partial z_{1[1,0]}}{\partial x_{[1,0]}} + \frac{\partial J}{\partial z_{1[1,1]}} \cdot \frac{\partial z_{1[1,1]}}{\partial x_{[1,0]}} = \frac{\partial J}{\partial z_{1[1,0]}} k_{[0,1]} + \frac{\partial J}{\partial z_{1[1,1]}} k_{[0,0]} \tag{61}
$$

Then the general form for compute $\frac{\partial J}{\partial X}$ is shown in equation (62)

$$\frac{\partial J}{\partial X} = \frac{\partial J}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial X} \rightarrow \frac{\partial J}{\partial x_{i,j,k}} = \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} k_{p,q} \frac{\partial J}{\partial Z_{1,i-p,j-q}} \tag{62}$$

where $k_{p,q}$ is the weight parameter of the convolutional filter at position (p, q), and $Z_{1,i-p,j-q}$ is the output feature map of the convolutional layer at position $(i-p, j-q)$.

2) Multiple convolution layers, with ReLU and softmax: This architecture will be illustrated in Table 6

5. Conclusion

In conclusion, deep learning frameworks like TensorFlow and PyTorch provide researchers with a range of tools and resources that make it easier to build, test, and deploy deep learning models. However, relying solely on these frameworks can result in a superficial understanding of the intricate mathematical mechanics involved in deep learning, which can impede the optimization process and hinder the achievement of optimal performance in developing deep networks. This paper aims to bridge this gap by simplifying and clarifying the mechanics of deep learning networks, providing a deeper insight into the topic, and enabling researchers to gain a deeper understanding of how deep learning works, ultimately improving their ability to optimize and improve deep learning models. Through explaining the derivative of various methods and activation functions, this paper aims to remove obstacles and empower researchers to create their own frameworks with enhanced speed and efficiency.

References

[1] D. Learning, "Deep learning," *High-dimensional fuzzy clustering*, 2020.

[2] A. F. Rasheed, M. Zarkoosh, and S. S. Al-Azzawi, "Multi-cnn voting method for improved arabic handwritten digits classification," in *2023 9th International Conference on Computer and Communication Engineering (ICCCE)*. IEEE, 2023, pp. 205–210.

[3] ——, "The impact of feature selection on malware classification using chi-square and machine learning," in *2023 9th International Conference on Computer and Communication Engineering (ICCCE)*. IEEE, 2023, pp. 211–216.

[4] A. F. Rasheed, M. Zarkoosh, and F. Elia, "Enhancing graphical password authentication system with deep learning-based arabic digit recognition," *International Journal of Information Technology*, pp. 1–9, 2023.

[5] A. F. Rasheed, M. Zarkoosh, S. F. Abbas, and S. Sabah Al-Azzawi, "Arabic offensive language classification: Leveraging transformer, lstm, and svm," in *2023 IEEE International Conference on Machine Learning and Applied Network Technologies (ICMLANT)*. IEEE, 2023, pp. 1–6.

[6] P. P. Shinde and S. Shah, "A review of machine learning and deep learning applications," in *2018 Fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE, 2018, pp. 1–6.

[7] H. Bolhasani, M. Mohseni, and A. M. Rahmani, "Deep learning applications for iot in health care: A systematic review," *Informatics in Medicine Unlocked*, vol. 23, p. 100550, 2021.

[8] Y. Bahri, J. Kadmon, J. Pennington, S. S. Schoenholz, J. Sohl-Dickstein, and S. Ganguli, "Statistical mechanics of deep learning," *Annual Review of Condensed Matter Physics*, vol. 11, pp. 501–528, 2020.

[9] A. S. Rajawat and S. Jain, "Fusion deep learning based on back propagation neural network for personalization," in *2nd International Conference on Data, Engineering and Applications (IDEA)*. IEEE, 2020, pp. 1–7.

[10] A. Parvat, J. Chavan, S. Kadam, S. Dev, and V. Pathak, "A survey of deep-learning frameworks," in *2017 International Conference on Inventive Systems and Control (ICISC)*. IEEE, 2017, pp. 1–7.

[11] H. Li, M. Krček, and G. Perin, "A comparison of weight initializers in deep learning-based side-channel analysis," in *Applied Cryptography and Network Security Workshops: ACNS 2020 Satellite Workshops, AIBlock, AIHWS, AIoTS, Cloud S&P, SCI, SecMT, and SiMLA, Rome, Italy, October 19–22, 2020, Proceedings 18*. Springer, 2020, pp. 126–143.

[12] Y. Cao, Z. Fang, Y. Wu, D.-X. Zhou, and Q. Gu, "Towards understanding the spec-

tral bias of deep learning," *arXiv preprint arXiv:1912.01198*, 2019.

[13] I. Garrido-Muñoz, A. Montejo-Ráez, F. Martínez-Santiago, and L. A. Ureña-López, "A survey on bias in deep nlp," *Applied Sciences*, vol. 11, no. 7, p. 3184, 2021.

[14] N. Papernot, A. Thakurta, S. Song, S. Chien, and Ú. Erlingsson, "Tempered sigmoid activations for deep learning with differential privacy," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 10, 2021, pp. 9312–9321.

[15] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[16] S. R. Dubey, S. K. Singh, and B. B. Chaudhuri, "Activation functions in deep learning: A comprehensive survey and benchmark," *Neurocomputing*, 2022.

[17] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[18] M. Uzair and N. Jamil, "Effects of hidden layers on the efficiency of neural networks," in *2020 IEEE 23rd international multitopic conference (INMIC)*. IEEE, 2020, pp. 1–6.

[19] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: analysis, applications, and prospects," *IEEE transactions on neural networks and learning systems*, 2021.

[20] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[21] J. Wu, "Introduction to convolutional neural networks," *National Key Lab for Novel Software Technology. Nanjing University. China*, vol. 5, no. 23, p. 495, 2017.

[22] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[23] J. Zhang, "Gradient descent based optimization algorithms for deep learning models training," *arXiv preprint arXiv:1903.03614*, 2019.

[24] N. Kriegeskorte and T. Golan, "Neural network models and deep learning," *Current Biology*, vol. 29, no. 7, pp. R231–R236, 2019.

[25] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," *Advances in neural information processing systems*, vol. 31, 2018.

[26] M. H. Sazli, "A brief review of feed-forward neural networks," *Communications Faculty of Sciences University of Ankara Series A2-A3 Physical Sciences and Engineering*, vol. 50, no. 01, 2006.

[27] J. Li, J.-h. Cheng, J.-y. Shi, and F. Huang, "Brief introduction of back propagation (bp) neural network algorithm and its improvement," in *Advances in Computer Science and Information Engineering: Volume 2*. Springer, 2012, pp. 553–558.

[28] I. Goodfellow, Y. Bengio, and A. Courville, "Regularization for deep learning," *Deep learning*, pp. 216–261, 2016.

[29] N. Srivastava, "Improving neural networks with dropout," *University of Toronto*, vol. 182, no. 566, p. 7, 2013.

[30] J. Ba and B. Frey, "Adaptive dropout for training deep neural networks," *Advances in neural information processing systems*, vol. 26, 2013.

[31] R. Courant, F. John, A. A. Blank, and A. Solomon, *Introduction to calculus and analysis*. Springer, 1965, vol. 1.

Table 4. Forward and Backward Equations for a Multilayer Perceptron with RelU and Sigmoid Activation Functions

Layer Name	Forward Equation	Backward Equation
Layer 1	$w^{[1]}_{current}$	$W^{[1]}_{new} := W^{[1]}_{current} - \alpha \frac{\partial J}{\partial W^{[1]}_{current}}$
		$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot X$
	$b^{[1]}_{current}$	$b^{[1]}_{new} := b^{[1]}_{current} - \alpha \frac{\partial J}{\partial b^{[1]}_{current}}$
		$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot 1$
	$z^{[1]} = XW^{[1]} + b^{[1]}$	$\frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[1]} = \begin{cases} Z^{[1]}, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot W^{[2]}$
Layer 2	$w^{[2]}_{current}$	$W^{[2]}_{new} := W^{[2]}_{current} - \alpha \frac{\partial J}{\partial W^{[2]}_{current}}$
		$\frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial W^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot (A^{[2]})^T$
	$b^{[2]}_{current}$	$b^{[2]}_{new} := b^{[2]}_{current} - \alpha \frac{\partial J}{\partial b^{[2]}_{current}}$
		$\frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot 1$
	$Z^{[2]} = A^{[1]}W^{[2]} + b^{[2]}$	$\frac{\partial J}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot (\begin{cases} 1, & \text{if } Z^{[2]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[2]} = \begin{cases} Z^{[2]}, & \text{if } Z^{[2]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[2]}} = \frac{\partial J}{\partial Z^{[3]}} \cdot \frac{\partial Z^{[3]}}{\partial A^{[2]}} = \frac{\partial J}{\partial Z^{[3]}} \cdot W^{[3]}$
...	...	...
Layer L-1	$w^{[L-1]}_{current}$	$W^{[L-1]}_{new} := W^{[L-1]}_{current} - \alpha \frac{\partial J}{\partial W^{[L-1]}_{current}}$
		$\frac{\partial J}{\partial W^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot \frac{\partial Z^{[L-1]}}{\partial W^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot (A^{[L-1]})^T$
	$b^{[L-1]}_{current}$	$b^{[L-1]}_{new} := b^{[L-1]}_{current} - \alpha \frac{\partial J}{\partial b^{[L-1]}_{current}}$
		$\frac{\partial J}{\partial b^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot \frac{\partial Z^{[L-1]}}{\partial b^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot 1$
	$Z^{[L-1]} = A^{[L-2]}W^{[L-1]} + b^{[L-1]}$	$\frac{\partial J}{\partial Z^{[L-1]}} = \frac{\partial J}{\partial A^{[L-1]}} \cdot \frac{\partial A^{[L-1]}}{\partial Z^{[L-1]}} = \frac{\partial J}{\partial A^{[L-1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[L-1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[L-1]} = \begin{cases} Z^{[L-1]}, & \text{if } Z^{[L-1]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[L-1]}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial A^{[L-1]}} = \frac{\partial J}{\partial Z^{[L]}} \cdot W^{[L]}$
Layer L	$w^{[L]}_{current}$	$W^{[L]}_{new} := W^{[L]}_{current} - \alpha \frac{\partial J}{\partial W^{[L]}_{current}}$
		$\frac{\partial J}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot A^{[L]}$
	$b^{[L]}_{current}$	$b^{[L]}_{new} := b^{[L]}_{current} - \alpha \frac{\partial J}{\partial b^{[L]}_{current}}$
		$\frac{\partial J}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot 1$
	$z^{[L]} = XW^{[L]} + b^{[L]}$	$\frac{\partial J}{\partial Z^{[L]}} = \frac{\partial J}{\partial A^{[L]}} \cdot \frac{\partial A^{[L]}}{\partial Z^{[L]}} = \frac{\partial J}{\partial A^{[L]}} \cdot A^{[L]}(1 - A^{[L]})$
	$A^{[L]} = \frac{1}{1+e^{-Z^L}}$	$\frac{\partial J}{\partial A^{[L]}} = -\frac{Y}{A^{[L]}} + \frac{1-Y}{1-A^{[L]}}$
Loss Function	$J(A^{[L]}, Y) = Y \log(A^{[L]}) + (1 - Y) \log(1 - A^{[L]})$	

Table 5. Forward and Backward Equations for a Multilayer Perceptron with RelU and Softmax Activation Functions

Layer Name	Forward Equation	Backward Equation
Layer 1	$w^{[1]}_{current}$	$W^{[1]}_{new} := W^{[1]}_{current} - \alpha \frac{\partial J}{\partial W^{[1]}_{current}}$
		$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial W^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot X$
	$b^{[1]}_{current}$	$b^{[1]}_{new} := b^{[1]}_{current} - \alpha \frac{\partial J}{\partial b^{[1]}_{current}}$
		$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot 1$
	$z^{[1]} = XW^{[1]} + b^{[1]}$	$\frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[1]} = \begin{cases} Z^{[1]}, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial A^{[1]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot W^{[2]}$
...	...	...
Layer L	$w^{[L]}_{current}$	$W^{[L]}_{new} := W^{[L]}_{current} - \alpha \frac{\partial J}{\partial W^{[L]}_{current}}$
		$\frac{\partial J}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot A^{[L]}$
	$b^{[L]}_{current}$	$b^{[L]}_{new} := b^{[L]}_{current} - \alpha \frac{\partial J}{\partial b^{[L]}_{current}}$
		$\frac{\partial J}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot 1$
	$Z^{[L]} = XW^{[L]} + b^{[L]}$	$\frac{\partial J}{\partial Z^{[L]}} = \frac{\partial J}{\partial A^{[L]}} \cdot \frac{\partial A^{[L]}}{\partial Z^{[L]}} = A^L - Y$
	$A = \begin{bmatrix} \frac{e^{z^L_1}}{\sum_i^C e^{z^L_i}} \\ \frac{e^{z^L_2}}{\sum_i^C e^{z^L_i}} \\ \vdots \\ \frac{e^{z^L_C}}{\sum_i^C e^{z_i}} \end{bmatrix}$	$\frac{\partial J}{\partial A^{[L]}} = -\sum_{i=1}^C y_i \log(a^L_i)$
Loss Function	$L(Y, A^{[L]}) = -\sum_{i=1}^C y_i \log(a^L_i)$	

Table 6. Multiple convolution layers, with ReLU and softmax

Layer Name	Forward Equation	Backward Equation
Convolution 1	$K^{[1]}_{current}$	$K^{[1]}_{new} := K^{[1]}_{current} - \alpha \frac{\partial J}{\partial K^{[1]}_{current}}$
		$\frac{\partial J}{\partial K^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial K^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot X$
	$b^{[1]}_{current}$	$b^{[1]}_{new} := b^{[1]}_{current} - \alpha \frac{\partial J}{\partial b^{[1]}_{current}}$
		$\frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot \frac{\partial Z^{[1]}}{\partial b^{[1]}} = \frac{\partial J}{\partial Z^{[1]}} \cdot 1$
	$Z^{[1]} = X * K^{[1]} + b^{[1]}$	$\frac{\partial J}{\partial Z^{[1]}} = \frac{\partial J}{\partial A^{[1]}} \cdot \frac{\partial A^{[1]}}{\partial Z^{[1]}} =$
		$\frac{\partial J}{\partial A^{[1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[1]} = \begin{cases} Z^{[1]}, & \text{if } Z^{[1]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[1]}} = \frac{\partial J}{\partial Y^{[1]}_{pool}} \cdot \frac{\partial Y^{[1]}_{pool}}{\partial A^{[1]}} = \frac{\partial J}{\partial Y^{[1]}_{pool}} \cdot M^{[1]}$
Max pooling 1	$Y^1_{pool} = \text{Maxpooling}(A^1)$	$\frac{\partial J}{\partial Y^1_{pool}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial Y^1_{pool}} = \frac{\partial J}{\partial Z^{[2]}} \cdot K^{[2]}$
Convolution 2	$K^{[2]}_{current}$	$K^{[2]}_{new} := K^{[2]}_{current} - \alpha \frac{\partial J}{\partial K^{[2]}_{current}}$
		$\frac{\partial J}{\partial K^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial K^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot Y^{[1]}_{pool}$
	$b^{[2]}_{current}$	$b^{[2]}_{new} := b^{[2]}_{current} - \alpha \frac{\partial J}{\partial b^{[2]}_{current}}$
		$\frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot \frac{\partial Z^{[2]}}{\partial b^{[2]}} = \frac{\partial J}{\partial Z^{[2]}} \cdot 1$
	$Z^{[2]} = Y^{[1]}_{pool} * K^{[2]} + b^{[2]}$	$\frac{\partial J}{\partial Z^{[2]}} = \frac{\partial J}{\partial A^{[2]}} \cdot \frac{\partial A^{[2]}}{\partial Z^{[2]}} =$
		$\frac{\partial J}{\partial A^{[2]}} \cdot (\begin{cases} 1, & \text{if } Z^{[2]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[2]} = \begin{cases} Z^{[2]}, & \text{if } Z^{[2]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[2]}} = \frac{\partial J}{\partial Y^{[2]}_{pool}} \cdot \frac{\partial Y^{[2]}_{pool}}{\partial A^{[2]}} = \frac{\partial J}{\partial Y^{[2]}_{pool}} \cdot M^{[2]}$
Max pooling 2	$Y^2_{pool} = \text{Maxpooling}(A^{[2]})$	$\frac{\partial J}{\partial Y^2_{pool}} = \frac{\partial J}{\partial Z^{[3]}} \cdot \frac{\partial Z^3}{\partial Y^2_{pool}} = \frac{\partial J}{\partial Z^{[3]}} \cdot W^{[3]}$
...	...	...
Layer L-1 Deep	$W^{[L-1]}_{current}$	$W^{[L-1]}_{new} := W^{[L-1]}_{current} - \alpha \frac{\partial J}{\partial W^{[L-1]}_{current}}$
		$\frac{\partial J}{\partial W^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot \frac{\partial Z^{[L-1]}}{\partial W^{[L-1]}} =$
		$\frac{\partial J}{\partial Z^{[L-1]}} \cdot Y^{[L-1]}_{pool}$
	$b^{[L-1]}_{current}$	$b^{[L-1]}_{new} := b^{[L-1]}_{current} - \alpha \frac{\partial J}{\partial b^{[L-1]}_{current}}$
		$\frac{\partial J}{\partial b^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot \frac{\partial Z^{[L-1]}}{\partial b^{[L-1]}} = \frac{\partial J}{\partial Z^{[L-1]}} \cdot 1$
	$Z^{[L-1]} = Y^{[L-2]}_{pool} W^{[L-1]} + b^{[L-1]}$	$\frac{\partial J}{\partial Z^{[L-1]}} = \frac{\partial J}{\partial A^{[L-1]}} \cdot \frac{\partial A^{[L-1]}}{\partial Z^{[L-1]}} =$
		$\frac{\partial J}{\partial A^{[L-1]}} \cdot (\begin{cases} 1, & \text{if } Z^{[L-1]} > 0 \\ 0, & \text{otherwise} \end{cases})$
	$A^{[L-1]} = \begin{cases} Z^{[L-1]}, & \text{if } Z^{[L-1]} > 0 \\ 0, & \text{otherwise} \end{cases}$	$\frac{\partial J}{\partial A^{[L-1]}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial A^{[L-1]}} = \frac{\partial J}{\partial Z^{[L]}} \cdot W^{[L]}$
Layer L	$W^{[L]}_{current}$	$W^{[L]}_{new} := W^{[L]}_{current} - \alpha \frac{\partial J}{\partial W^{[L]}_{current}}$
		$\frac{\partial J}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial W^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot A^{[L]}$
	$b^{[L]}_{current}$	$b^{[L]}_{new} := b^{[L]}_{current} - \alpha \frac{\partial J}{\partial b^{[L]}_{current}}$
		$\frac{\partial J}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot \frac{\partial Z^{[L]}}{\partial b^{[L]}_{current}} = \frac{\partial J}{\partial Z^{[L]}} \cdot 1$
	$Z^{[L]} = A^{[L-1]} W^{[L]} + b^{[L]}$	$\frac{\partial J}{\partial Z^{[L]}} = \frac{\partial J}{\partial A^{[L]}} \cdot \frac{\partial A^{[L]}}{\partial Z^{[L]}} = A^L - Y$
	$A^{[L]} = \begin{bmatrix} \frac{e^{z^L_1}}{\sum_i^C e^{z^L_i}} \\ \frac{e^{z^L_2}}{\sum_i^C e^{z^L_i}} \\ \vdots \\ \frac{e^{z^L_C}}{\sum_i^C e^{z_i}} \end{bmatrix}$	$\frac{\partial J}{\partial A^{[L]}} = -\sum_{i=1}^C y_i \log(a^L_i)$
Loss Function	$L(Y, A^{[L]}) = -\sum_{i=1}^C y_i \log(a^L_i)$	