Homework part 2 - 99222084

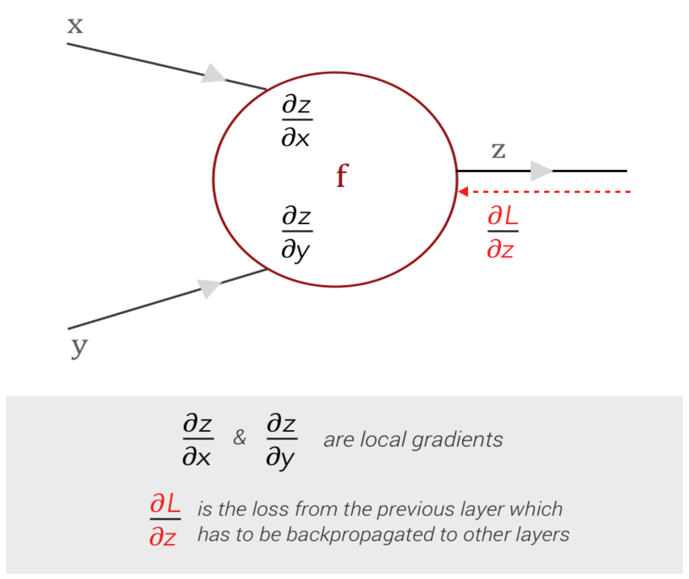
1-Describe the backpropagation details in the convolutional layers.

Simple explanation of backpropagation:

First, it's important to understand how backpropagation works. Backpropagation, or backward error propagation, is an algorithm for supervised learning in neural networks using gradient descent. The idea is that gradients are calculated in reverse order throughout the network. The gradient for the output layer weights is computed first, and the gradient for the input layer comes last. The gradient of each layer is computed using the partial derivatives of the gradient from the next layer. This backward flow of error information allows efficient gradient calculation for each layer compared to calculating each layer's gradient separately.

Chain rule in convolutional layer:

consider this graph:

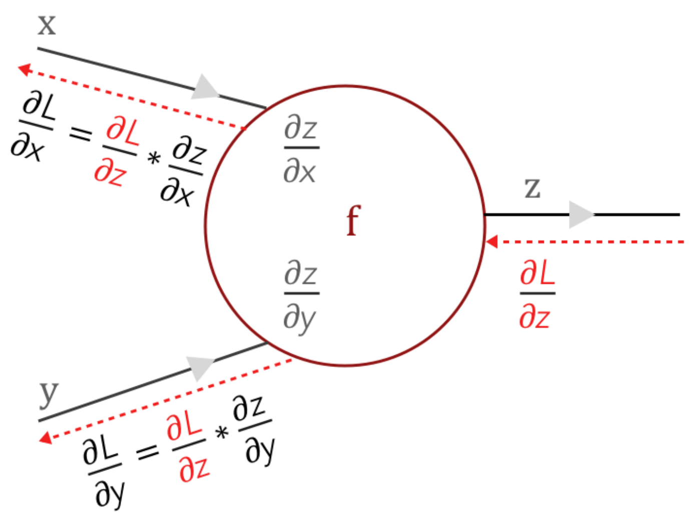

x
y
f
z
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$

$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

Here, we move forward through the entire CNN, and at the end, we calculate the loss using the loss function.

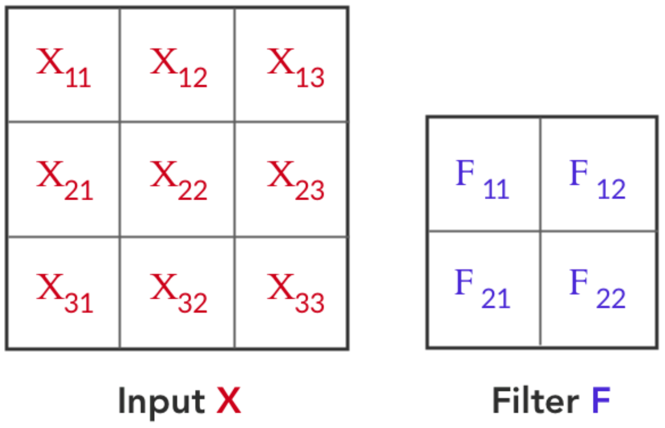
When we start calculating the loss during backpropagation, we reduce the gradient of the loss from the previous layer. Then, to propagate the loss to other gates, we need to compute ∂L/∂x and ∂L/∂y from ∂L/∂z. The chain rule helps in calculating these two derivatives.


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} * \frac{\partial z}{\partial y}$

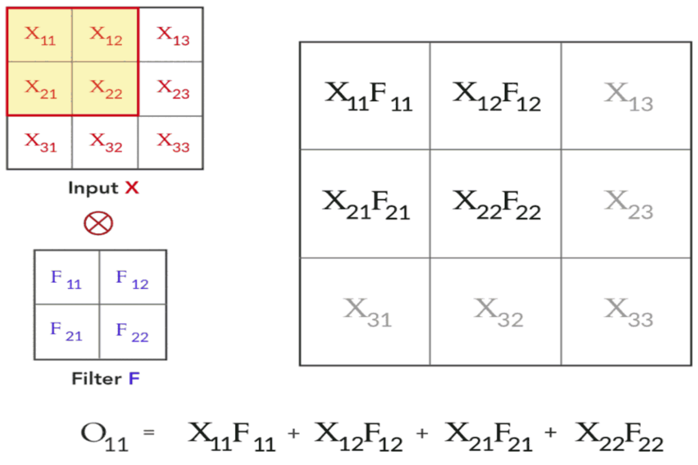
$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

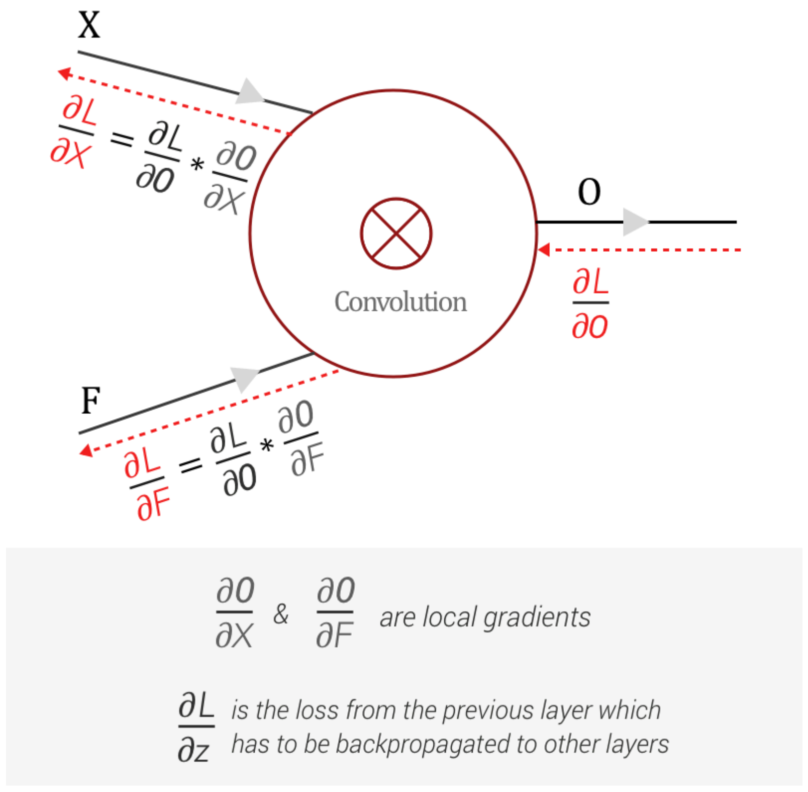
Now, let's assume this function becomes a convolutional function, and f represents our filter, which we consider as follows:



Which ultimately will be the final output of our convolutional function:



Which gives us this forward pass. As mentioned earlier, we obtain the gradient of the loss with respect to the output of each layer using ∂L/∂O, passing it backward through the layers. Thus, for the backward pass, we will have:


X
$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial X}$
O
Convolution
$\frac{\partial L}{\partial O}$
F
$\frac{\partial L}{\partial F} = \frac{\partial L}{\partial O} * \frac{\partial O}{\partial F}$

$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

Now, we can calculate ∂L/∂x and ∂L/∂F.

In the first step, we calculate ∂O/∂F. This means we need to subtract the output from the filter f. So, we subtract each output individually from its corresponding filter, one by one:

Local Gradients $\longrightarrow \textcircled{A}$

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Finding derivatives with respect to F_{11} , F_{12} , F_{21} and F_{22}

$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$$

Similarly, we can find the local gradients for O_{12} , O_{21} and O_{22}

:

In the second step, using the chain rule, we will have:

For every element of F

$$\frac{\partial L}{\partial F_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial F_i}$$

Expanding it, we will have:

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{11}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{11}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{11}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{11}}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{12}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{12}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{12}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{12}}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{21}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{21}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{21}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{21}}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * \frac{\partial O_{11}}{\partial F_{22}} + \frac{\partial L}{\partial O_{12}} * \frac{\partial O_{12}}{\partial F_{22}} + \frac{\partial L}{\partial O_{21}} * \frac{\partial O_{21}}{\partial F_{22}} + \frac{\partial L}{\partial O_{22}} * \frac{\partial O_{22}}{\partial F_{22}}$$

And by substituting the values, we will have:

$$\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial O_{11}} * X_{11} + \frac{\partial L}{\partial O_{12}} * X_{12} + \frac{\partial L}{\partial O_{21}} * X_{21} + \frac{\partial L}{\partial O_{22}} * X_{22}$$

$$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial O_{11}} * X_{12} + \frac{\partial L}{\partial O_{12}} * X_{13} + \frac{\partial L}{\partial O_{21}} * X_{22} + \frac{\partial L}{\partial O_{22}} * X_{23}$$

$$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial O_{11}} * X_{21} + \frac{\partial L}{\partial O_{12}} * X_{22} + \frac{\partial L}{\partial O_{21}} * X_{31} + \frac{\partial L}{\partial O_{22}} * X_{32}$$

$$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial O_{11}} * X_{22} + \frac{\partial L}{\partial O_{12}} * X_{23} + \frac{\partial L}{\partial O_{21}} * X_{32} + \frac{\partial L}{\partial O_{22}} * X_{33}$$

If we look closely, we see that ∂L/∂F is nothing but the convolution between the input and the gradient of the loss from the previous layer.

To obtain ∂L/∂x, we first calculate ∂O/∂x.

Local Gradients: $\longrightarrow$ B

$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Differentiating with respect to X_{11}, X_{12}, X_{21} *and* X_{22}

$$\frac{\partial O_{11}}{\partial X_{11}} = F_{11} \quad \frac{\partial O_{11}}{\partial X_{12}} = F_{12} \quad \frac{\partial O_{11}}{\partial X_{21}} = F_{21} \quad \frac{\partial O_{11}}{\partial X_{22}} = F_{22}$$

Similarly, we can find local gradients for O_{12}, O_{21} *and* O_{22}

Using the chain rule, we will have:

For every element of X_i

$$\frac{\partial L}{\partial X_i} = \sum_{k=1}^{M} \frac{\partial L}{\partial O_k} * \frac{\partial O_k}{\partial X_i}$$

When we expand it, we get:

$$\frac{\partial L}{\partial X_{11}} = \frac{\partial L}{\partial O_{11}} * F_{11}$$

$$\frac{\partial L}{\partial X_{12}} = \frac{\partial L}{\partial O_{11}} * F_{12} + \frac{\partial L}{\partial O_{12}} * F_{11}$$

$$\frac{\partial L}{\partial X_{13}} = \frac{\partial L}{\partial O_{12}} * F_{12}$$

$$\frac{\partial L}{\partial X_{21}} = \frac{\partial L}{\partial O_{11}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{11}$$

$$\frac{\partial L}{\partial X_{22}} = \frac{\partial L}{\partial O_{11}} * F_{22} + \frac{\partial L}{\partial O_{12}} * F_{21} + \frac{\partial L}{\partial O_{21}} * F_{12} + \frac{\partial L}{\partial O_{22}} * F_{11}$$

$$\frac{\partial L}{\partial X_{23}} = \frac{\partial L}{\partial O_{12}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{12}$$

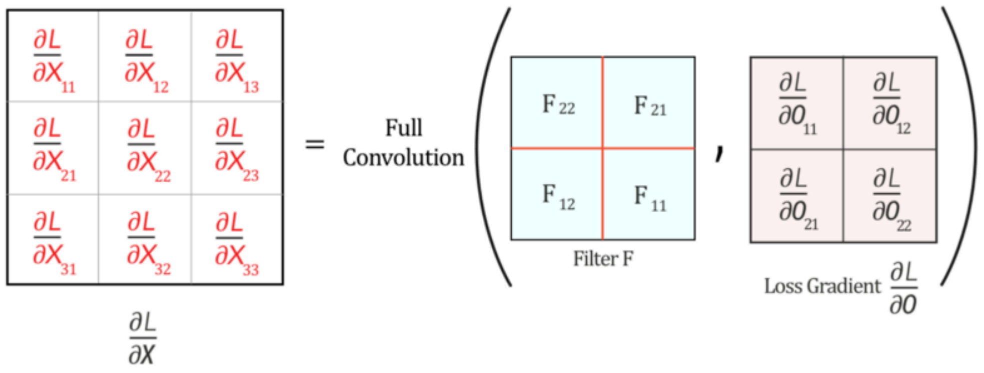
$$\frac{\partial L}{\partial X_{31}} = \frac{\partial L}{\partial O_{21}} * F_{21}$$

$$\frac{\partial L}{\partial X_{32}} = \frac{\partial L}{\partial O_{21}} * F_{22} + \frac{\partial L}{\partial O_{22}} * F_{21}$$

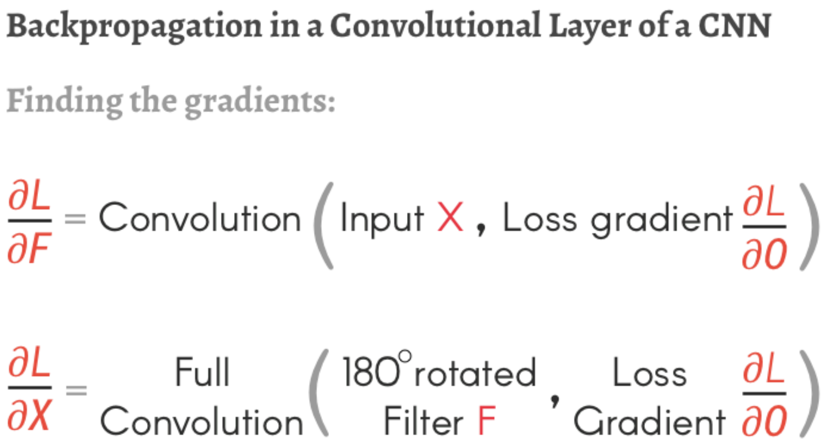
$$\frac{\partial L}{\partial X_{33}} = \frac{\partial L}{\partial O_{22}} * F_{22}$$

In the end, we will have ∂L/∂x. This operation is essentially the convolution operation.

Since full convolution generates ∂L/∂x, we have:


$\frac{\partial L}{\partial X}$
$\frac{\partial L}{\partial X_{11}}$ $\frac{\partial L}{\partial X_{12}}$ $\frac{\partial L}{\partial X_{13}}$
$\frac{\partial L}{\partial X_{21}}$ $\frac{\partial L}{\partial X_{22}}$ $\frac{\partial L}{\partial X_{23}}$
$\frac{\partial L}{\partial X_{31}}$ $\frac{\partial L}{\partial X_{32}}$ $\frac{\partial L}{\partial X_{33}}$
= Full Convolution
F 22 F 21 F 12 F 11
Filter F
$\frac{\partial L}{\partial O_{11}}$ $\frac{\partial L}{\partial O_{12}}$ $\frac{\partial L}{\partial O_{21}}$ $\frac{\partial L}{\partial O_{22}}$
Loss Gradient $\frac{\partial L}{\partial O}$

Now we have both ∂L/∂F and ∂L/∂x. The final result will show that both backpropagation and forward convolution are involved:



Backpropagation in a Convolutional Layer of a CNN

Finding the gradients:

$$\frac{\partial L}{\partial F} = \text{Convolution}\left(\text{Input } X,\ \text{Loss gradient } \frac{\partial L}{\partial O}\right)$$

$$\frac{\partial L}{\partial X} = \text{Full Convolution}\left(180^{\circ}\text{rotated Filter } F,\ \text{Loss Gradient } \frac{\partial L}{\partial O}\right)$$

What is the symmetry breaking phenomenon in artificial neural networks? What can be done to prevent this from happening?

In neural networks, the models we design should be such that they first receive data as input, process it through hidden layers, and then output the results. Ultimately, backpropagation should update the weights to refine the model. The initialization of weights has a significant impact on the learning process and model improvement. The simplest way to initialize weights is to set them all to zero, which can lead to the problem of "dead neurons."

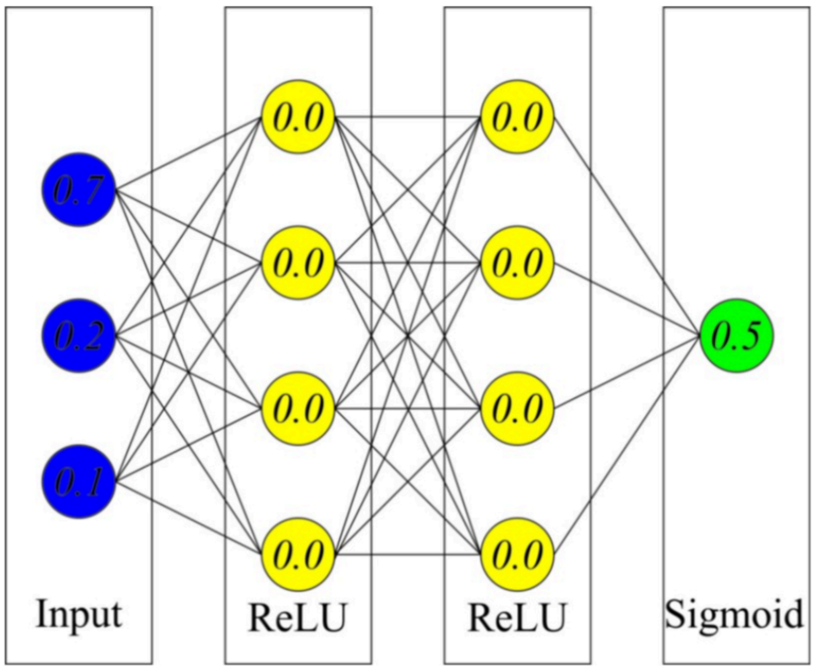


Image by author

To solve this problem, one can set each neuron's bias to one. This approach will certainly change the weights because ReLU neurons will produce non-zero outputs, but the changes will be suboptimal. Moreover, each neuron in the same layer will exhibit similar behavior and weights. This phenomenon is called the symmetry problem.

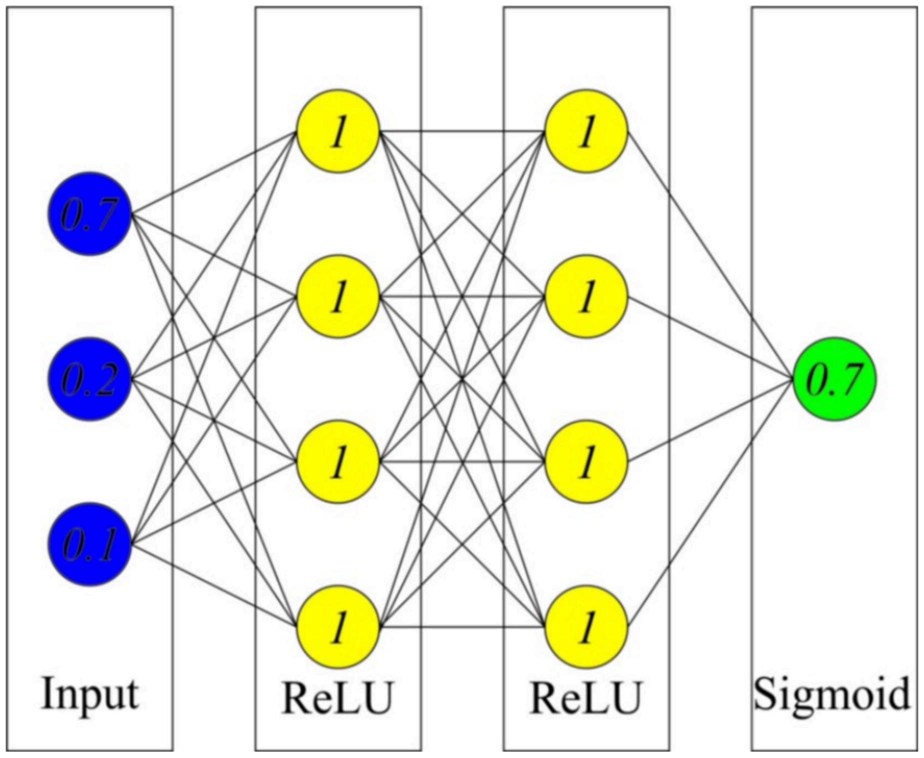
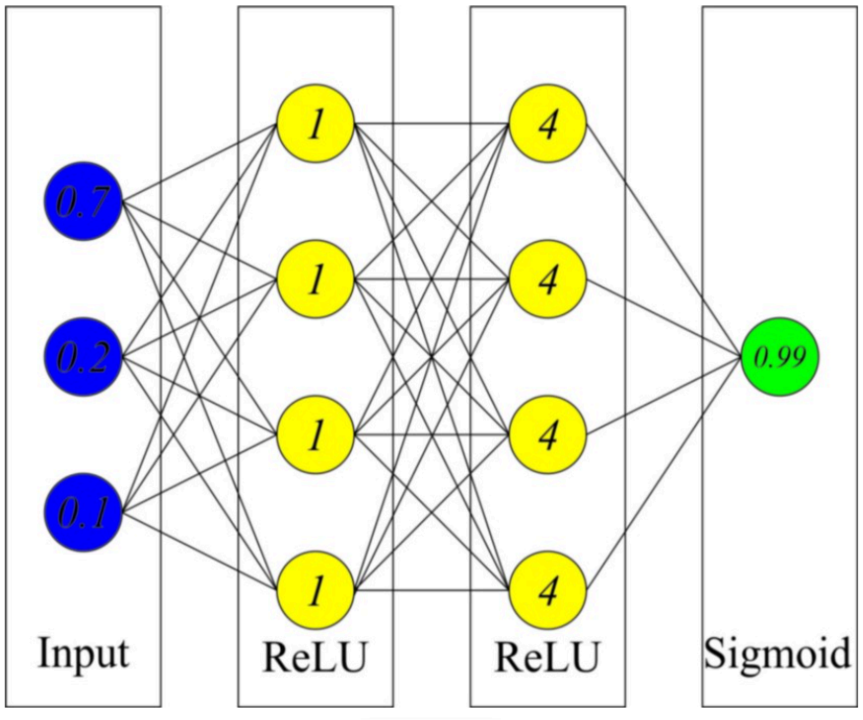
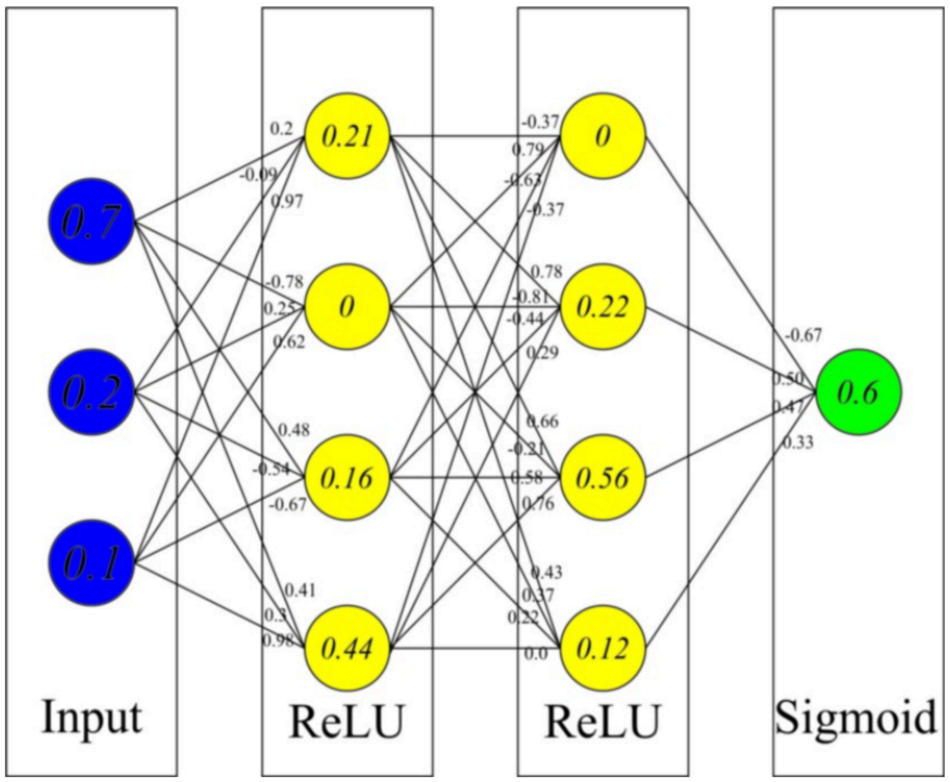


Image by author

Even weight initialization does not resolve this issue.



If all weights are the same, it's problematic because it means each neuron in a layer represents the same feature. Adding more neurons in a layer does not increase the complexity of the neural network, as the layer effectively behaves like just one neuron. The simple solution to this problem is to initialize weights randomly. For example, randomize the weights and set the biases to zero.



Random initialization allows us to break this symmetry, enabling each neuron in the neural network to behave differently.

Random Normal Initialization: To have a specific range, weights can be initialized with a random normal distribution. Instead of initializing weights with just one random number, using a normal distribution allows us to set a mean and standard deviation to initialize weights within a certain range. For example, setting the mean to zero and the standard deviation to one gives a range approximately between -1 and 1.

Xavier Initialization: A better technique for initializing neural networks is to control the output variance. We want the output of a layer produced by neurons to follow the same distribution. Xavier initialization is a technique that initializes weights so that the outputs produced by neurons follow the same distribution.

What are the benefits of the pooling layers? What are the drawbacks of the pooling layers? Are you willing to use these layers? Can you use these layers frequently?

Pooling layers are used to reduce the dimensions of feature maps. This decreases the number of learning parameters and the computational load of the network. Pooling layers summarize features in a region of the feature map created by a convolutional layer. Therefore, operations are performed on summarized features instead of the exact features produced by the convolutional layer. This makes the model more robust to changes in the position of features in the input image.

Although max pooling helps increase the variability in position and lighting conditions, it reduces the resolution of the feature map and can cause the loss of information about distinguishing features if most pixels in a pooling region have high pooling values.

Pooling, if not used excessively, can be very useful and practical because, as mentioned, it might cause the loss of critical information. On the other hand, pooling provides a smaller representation of the central details of the image, which might reduce the attention and importance of the main parts of the image. Proper use of padding may not always be effective, so pooling should be applied thoughtfully and according to its applications and principles. Even today’s large models use pooling. Proper use of pooling layers depends not only on knowledge but also on experience.

Please describe the differences between the cross-entropy vs. quadratic cost. Which one do you prefer for the classification problem?

The learning rate changes differently with these two costs. Specifically, when using quadratic cost, learning is slower when a neuron is clearly wrong because the neuron approaches the correct output more gradually. In contrast, with cross-entropy, learning is faster when a neuron is clearly wrong.

Cross-entropy loss is used in classification tasks where we aim to minimize the probability of a negative class by maximizing the expected value of some function over the training data. The idea behind this loss function is to impose a high penalty for incorrect predictions and a low penalty for correct classifications.

There is an alternative activation function for ReLU called leaky ReLU. I would like you to compare them.

· Which one is faster?

The difference in appearance is minimal, but leaky ReLU tends to perform better during training with higher epochs because it does not increase data sparsity at zero like ReLU. It tends to approach zero with a very small and reasonable slope, which can improve performance. During training, it is faster due to better balance and might avoid some neurons turning into dead neurons with more iterations, leading to better model performance.

· Which one can prevent gradient vanishing, and how?

Leaky ReLU has two advantages over its simpler counterpart:

1. It effectively solves the dying ReLU problem.
2. It trains faster.

The near-zero "mean activation" speeds up training. Vanishing gradients occur when weights are not updated properly and become so small that they cannot be trained, causing the model to learn nothing. Since leaky ReLU does not zero out values for negative inputs but assigns them a small value instead, it does not create the same problem as ReLU.