Back Propagation in Convolutional Neural Networks — Intuition and Code

Mayank Agarwal Follow
Dec 14, 2017 · 4 min read

IT IS FRIDAY, ALL MY FRIENDS ARE AT PARTY
BUT, I'M HERE LEARNING BACK PROPAGATION
memegenerator.net

> *Disclaimer: If you don't have any idea of how back propagation operates on a computational graph, I recommend you have a look at this lecture from the famous cs231n course.*

I have scratched my head for a long time wondering how the back propagation algorithm works for convolutions. I could not find a simple and intuitive explanation of the algorithm online. So, I decided to write one myself. Hope you enjoy!

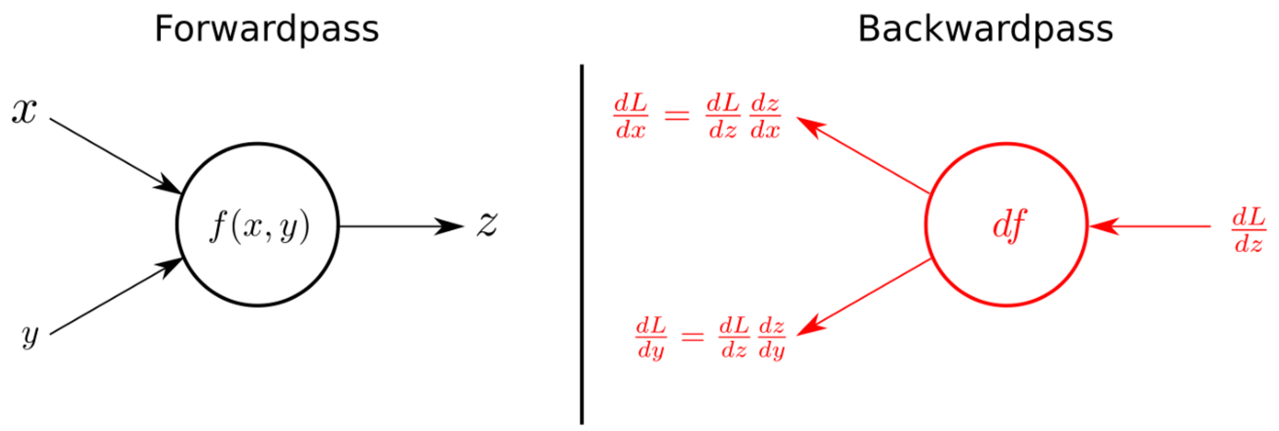
Why Understand Back Propagation?

Andrej Karpathy wrote in his *blog* about the need of understanding back propagation coining it as a *Leaky Abstraction*

> *"it is easy to fall into the trap of abstracting away the learning process — believing that you can simply stack arbitrary layers together and backprop will "magically make them work" on your data"*

The Chain Rule

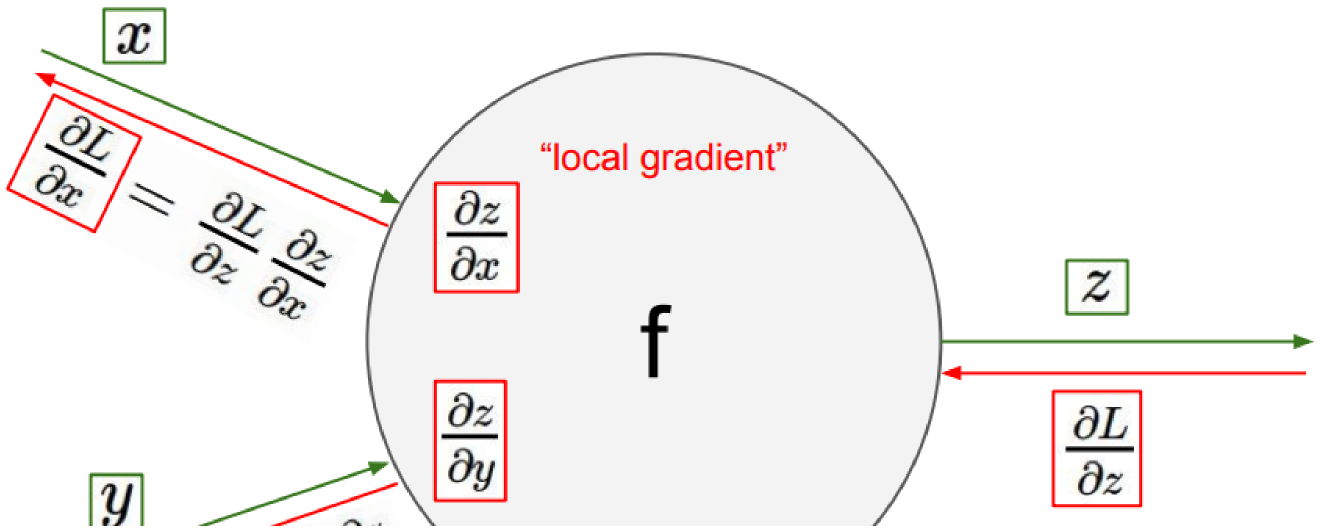
The following figure summarises the use of chain rule for the backward pass in computational graphs.



Forwardpass
Backwardpass
x
y
$f(x,y)$
z
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$

The forward pass on the left calculates **z** as a function **f(x,y)** using the input variables **x** and **y.** The right side of the figures shows the backward pass. Receiving **dL/dz**, the gradient of the loss function with respect to **z** from above, the gradients of **x** and **y** on the loss function can be calculate by applying the chain rule, as shown in the figure (borrowed from this post)

Here is another illustration which talks about the local gradients.



x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
y

$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
gradients

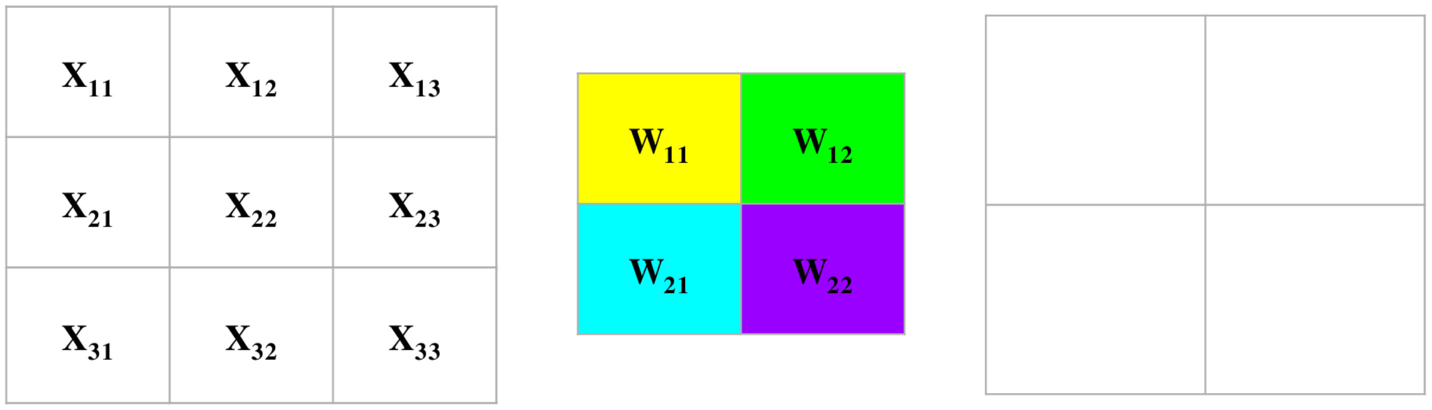
Back propagation illustration from CS231n Lecture 4. The variables **x** and **y** are cached, which are later used to calculate the local gradients.

If you understand the chain rule, you are good to go.

Let's Begin

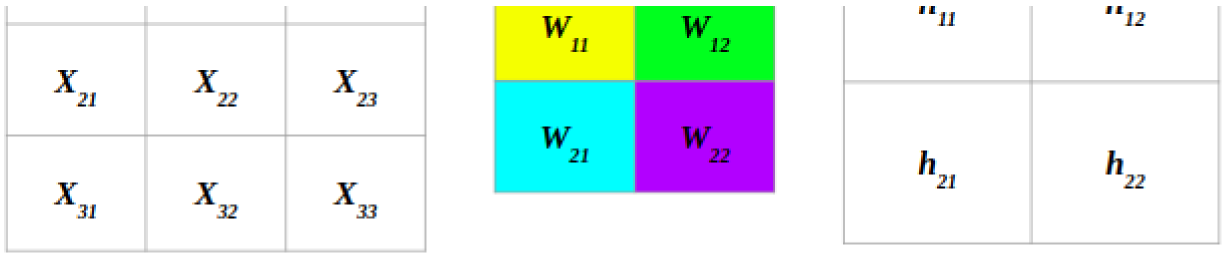
We will try to understand how the backward pass for a single convolutional layer by taking a simple case where number of channels is one across all computations. We will also dive into the code later.

The following convolution operation takes an input **X** of size **3x3** using a single filter **W** of size **2x2** without any padding and **stride = 1** generating an output **H** of size **2x2**. Also note that, while performing the forward pass, we will cache the variables **X** and filter **W**. This will help us while performing the backward pass.



Convolution Operation (Forward Pass)

> ***Note**: Here we are performing the convolution operation without flipping the filter. This is also referred to as the cross-correlation operation in literature. The above animation is provided just for the sake of clarity.*



Input Size : 3x3, **Filter Size** : 2x2, **Output Size** : 2x2

$$h_{11} = W_{11}X_{11} + W_{12}X_{12} + W_{21}X_{21} + W_{22}X_{22}$$

$$h_{12} = W_{11}X_{12} + W_{12}X_{13} + W_{21}X_{22} + W_{22}X_{23}$$

$$h_{21} = W_{11}X_{21} + W_{12}X_{22} + W_{21}X_{31} + W_{22}X_{32}$$

$$h_{22} = W_{11}X_{22} + W_{12}X_{23} + W_{21}X_{32} + W_{22}X_{33}$$

Output Equations

Backward Pass

Before moving further, make note of the following notations.

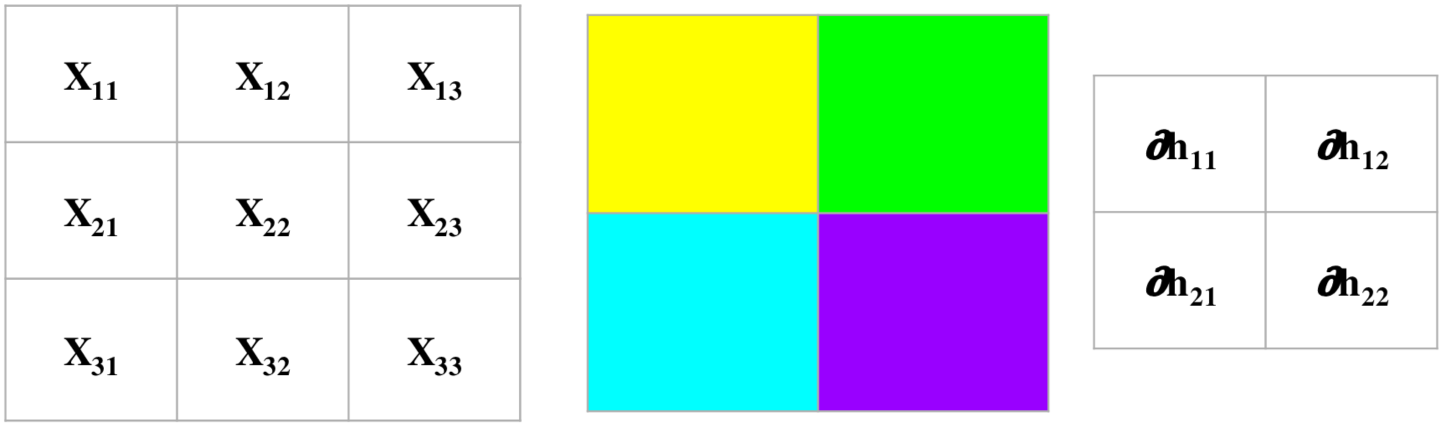
$$\partial h_{ij} \text{ represents } \frac{\partial L}{\partial h_{ij}}$$

$$\partial w_{ij} \text{ represents } \frac{\partial L}{\partial w_{ij}}$$

Notations

Now, for implementing the back propagation step for the current layer, we can assume that we get **∂h** as input (from the backward pass of the next layer) and our aim is to calculate **∂w** and **∂x**. It is important to understand that **∂x** (or **∂h** for previous layer) would be the input for the backward pass of the previous layer. This is the core principle behind the success of back propagation.

Each weight in the filter contributes to each pixel in the output map. Thus, any change in a weight in the filter will affect all the output pixels. Thus, all these changes add up to contribute to the final loss. Thus, we can easily calculate the derivatives as follows.



X_{11}	X_{12}	X_{13}
X_{21}	X_{22}	X_{23}
X_{31}	X_{32}	X_{33}

🟨	🟩
🟦	🟪

∂h_{11}	∂h_{12}
∂h_{21}	∂h_{22}

Derivative Computation (Backward pass) since pictures speak more than words

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

Final derivatives after performing back propagation

Similarly, we can derive ∂x. Moving further, let’s see some code.

> ***Note**: Much of the code is inspired from a programming assignment from the course Convolutional Neural Network by deeplearning.ai which is taught by Andrew Ng on Coursera.*

Naive implementation of forward and backward pass for a convolution function

```python
def conv_forward(X, W):
    '''
    The forward computation for a convolution function

    Arguments:
    X -- output activations of the previous layer, numpy array of shape (n_H_prev, n_W_p
    W -- Weights, numpy array of size (f, f) assuming number of filters = 1

    Returns:
    H -- conv output, numpy array of size (n_H, n_W)
    cache -- cache of values needed for conv_backward() function
    '''

    # Retrieving dimensions from X's shape
    (n_H_prev, n_W_prev) = X.shape

    # Retrieving dimensions from W's shape
    (f, f) = W.shape

    # Compute the output dimensions assuming no padding and stride = 1
    n_H = n_H_prev - f + 1
    n_W = n_W_prev - f + 1

    # Initialize the output H with zeros
    H = np.zeros((n_H, n_W))

    # Looping over vertical(h) and horizontal(w) axis of output volume
    for h in range(n_H):
        for w in range(n_W):
            x_slice = X[h:h+f, w:w+f]
            H[h,w] = np.sum(x_slice * W)

    # Saving information in 'cache' for backprop
    cache = (X, W)

    return H, cache
```

forward_pass.py hosted with ❤ by GitHub   view raw forward_pass.py hosted with ❤ by GitHub

```python
def conv_backward(dH, cache):
    '''
    The backward computation for a convolution function

    Arguments:
    dH -- gradient of the cost with respect to output of the conv layer (H), numpy array
    cache -- cache of values needed for the conv_backward(), output of conv_forward()

    Returns:
    dX -- gradient of the cost with respect to input of the conv layer (X), numpy array
    dW -- gradient of the cost with respect to the weights of the conv layer (W), numpy
    '''


    # Retrieving information from the "cache"
    (X, W) = cache

    # Retrieving dimensions from X's shape
    (n_H_prev, n_W_prev) = X.shape

    # Retrieving dimensions from W's shape
    (f, f) = W.shape

    # Retrieving dimensions from dH's shape
    (n_H, n_W) = dH.shape

    # Initializing dX, dW with the correct shapes
    dX = np.zeros(X.shape)
    dW = np.zeros(W.shape)

    # Looping over vertical(h) and horizontal(w) axis of the output
    for h in range(n_H):
        for w in range(n_W):
            dX[h:h+f, w:w+f] += W * dH(h,w)
            dW += X[h:h+f, w:w+f] * dH(h,w)

    return dX, dW
```

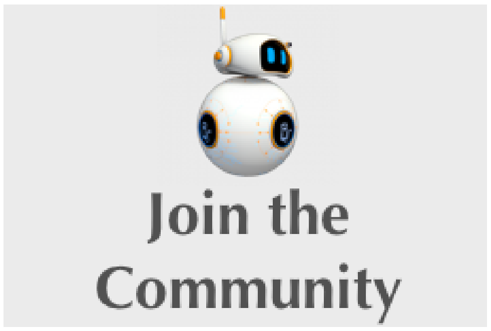
backward_pass.py hosted with ❤️ by GitHub view raw

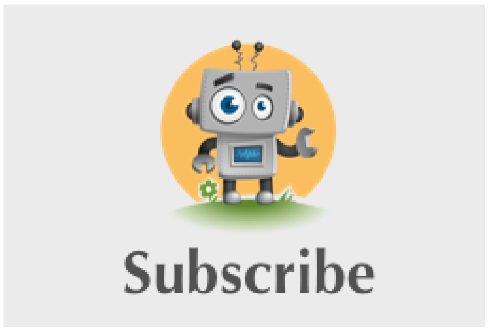
If you enjoyed this article, you might also want to check the following articles to delve deeper into mathematics:

- A Step by Step Backpropagation Example
- Derivation of Backpropagation in Convolutional Neural Network (CNN)
- Convolutional Neural Networks backpropagation: from intuition to derivation
- Backpropagation in Convolutional Neural Networks

I also found Back propagation in Convnets lecture by *Dhruv Batra* very useful for understanding the concept.

Since I might not be an expert on the topic, if you find any mistakes in the article, or have any suggestions for improvement, please mention in comments.





Deep Learning Convolution Neural Net Backpropagation Neural Networks

Artificial Intelligence