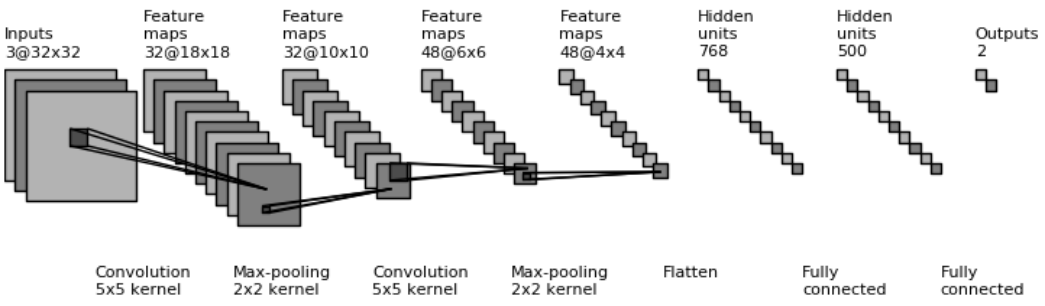
Backpropagation In Convolutional Neural Networks

Jefkine, 5 September 2016

Introduction

Convolutional neural networks (CNNs) are a biologically-inspired variation of the multilayer perceptrons (MLPs). Neurons in CNNs share weights unlike in MLPs where each neuron has a separate weight vector. This sharing of weights ends up reducing the overall number of trainable weights hence introducing sparsity.



Utilizing the weights sharing strategy, neurons are able to perform convolutions on the data with the convolution filter being formed by the weights. This is then

followed by a pooling operation which as a form of non-linear down-sampling, progressively reduces the spatial size of the representation thus reducing the amount of computation and parameters in the network.

Existing between the convolution and the pooling layer is an activation function such as the ReLu layer; a non-saturating activation is applied element-wise, i.e. $f(x) = max(0, x)$ thresholding at zero. After several convolutional and pooling layers, the image size (feature map size) is reduced and more complex features are extracted.

Eventually with a small enough feature map, the contents are squashed into a one dimension vector and fed into a fully-connected MLP for processing. The last layer of this fully-connected MLP seen as the output, is a loss layer which is used to specify how the network training penalizes the deviation between the predicted and true labels.

Before we begin lets take look at the mathematical definitions of convolution and cross-correlation:

Cross-correlation

Given an input image I and a filter (kernel) K of dimensions $k_1 \times k_2$, the cross-correlation operation is given by:

$$(I \otimes K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(m, n) \tag{1}$$

Convolution

Given an input image I and a filter (kernel) K of dimensions $k_1 \times k_2$, the convolution operation is given by:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m, n) \tag{2}$$

$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m, -n) \tag{3}$$

From Eq. 3 it is easy to see that convolution is the same as cross-correlation with a flipped kernel i.e: for a kernel K where $K(-m, -n) == K(m, n)$.

Convolution Neural Networks - CNNs

CNNs consists of convolutional layers which are characterized by an input map I, a bank of filters K and biases b.

In the case of images, we could have as input an image with height H, width W and $C = 3$ channels (red, blue and green) such that $I \in \mathbb{R}^{H \times W \times C}$. Subsequently for a bank of D filters we have $K \in \mathbb{R}^{k1 \times k2 \times C \times D}$ and biases $b \in \mathbb{R}^{D}$, one for each filter.

The output from this convolution procedure is as follows:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \sum_{c=1}^{C} K_{m,n,c} \cdot I_{i+m,j+n,c} + b \tag{4}$$

The convolution operation carried out here is the same as cross-correlation, except that the kernel is "flipped" (horizontally and vertically).

For the purposes of simplicity we shall use the case where the input image is grayscale i.e single channel $C = 1$. The Eq. 4 will be transformed to:

$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} K_{m,n} \cdot I_{i+m,j+n} + b \tag{5}$$

Notation

To help us explore the forward and backpropagation, we shall make use of the following notation:

1. l is the l^{th} layer where $l = 1$ is the first layer and $l = L$ is the last layer.
2. Input x is of dimension $H \times W$ and has i by j as the iterators
3. Filter or kernel w is of dimension $k_1 \times k_2$ has m by n as the iterators
4. $w^l_{m,n}$ is the weight matrix connecting neurons of layer l with neurons of layer $l - 1$.
5. b^l is the bias unit at layer l.
6. $x^l_{i,j}$ is the convolved input vector at layer l plus the bias represented as

$$x^l_{i,j} = \sum_m \sum_n w^l_{m,n} o^{l-1}_{i+m,j+n} + b^l$$

7. $o^l_{i,j}$ is the output vector at layer l given by

$$o^l_{i,j} = f(x^l_{i,j})$$

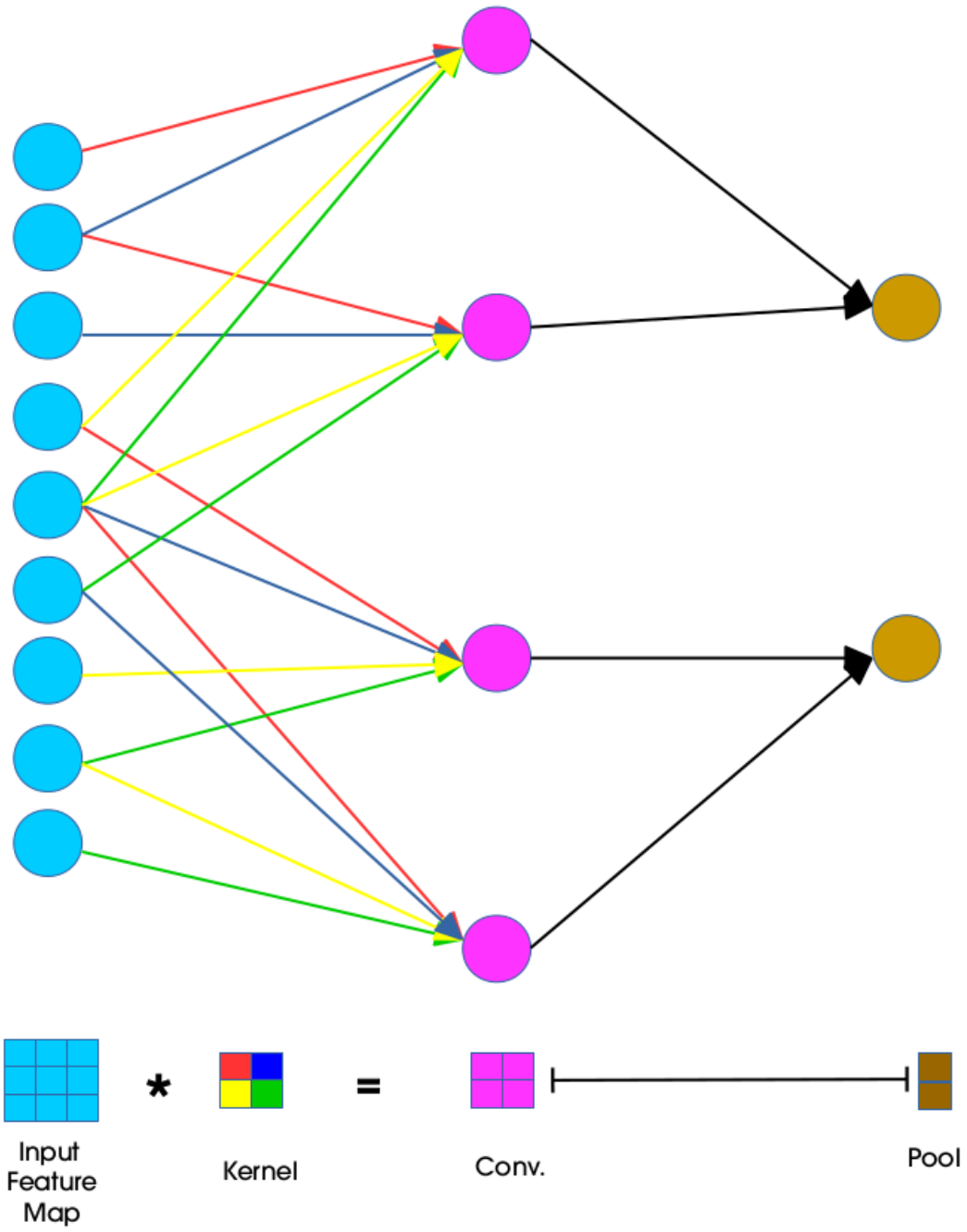
8. $f(\cdot)$ is the activation function. Application of the activation layer to the convolved input vector at layer l is given by $f(x^l_{i,j})$

Foward Propagation

To perform a convolution operation, the kernel is flipped 180° and slid across the input feature map in equal and finite strides. At each location, the product between

each element of the kernel and the input input feature map element it overlaps is computed and the results summed up to obtain the output at that current location.

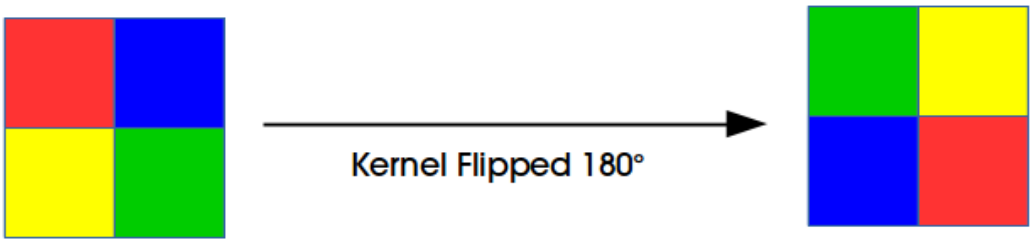
This procedure is repeated using different kernels to form as many output feature maps as desired. The concept of weight sharing is used as demonstrated in the diagram below:



Units in convolutional layer illustrated above have receptive fields of size 4 in the input feature map and are thus only connected to 4 adjacent neurons in the input layer. This is the idea of **sparse connectivity** in CNNs where there exists local connectivity pattern between neurons in adjacent layers.

The color codes of the weights joining the input layer to the convolutional layer show how the kernel weights are distributed (shared) amongst neurons in the adjacent layers. Weights of the same color are constrained to be identical.

The convolution process here is usually expressed as a cross-correlation but with a flipped kernel. In the diagram below we illustrate a kernel that has been flipped both horizontally and vertically:



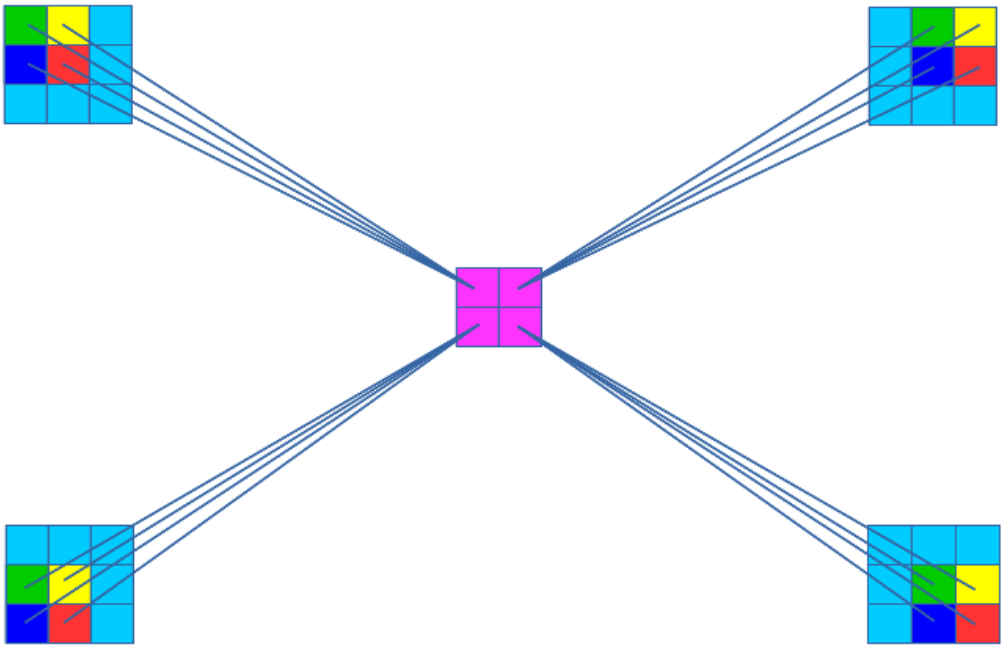
The convolution equation of the input at layer l is given by:

$$x_{i,j}^l = \text{rot}_{180^\circ}\left\{w_{m,n}^l\right\} * o_{i,j}^{l-1} + b_{i,j}^l \tag{6}$$

$$x_{i,j}^l = \sum_m \sum_n w_{m,n}^l o_{i+m,j+n}^{l-1} + b_{i,j}^l \tag{7}$$

$$o_{i,j}^l = f(x_{i,j}^l) \tag{8}$$

This is illustrated below:



Error

For a total of P predictions, the predicted network outputs y_p and their corresponding targeted values t_p the the mean squared error is given by:

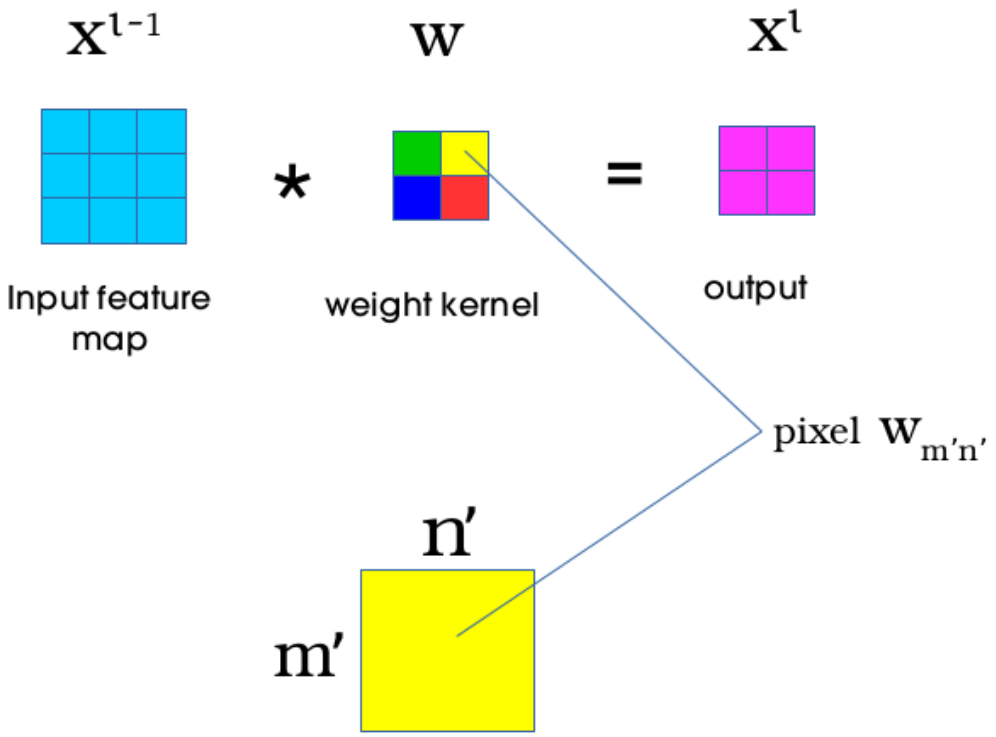
$$E = \frac{1}{2}\sum_p (t_p - y_p)^2 \tag{9}$$

Learning will be achieved by adjusting the weights such that y_p is as close as possible or equals to corresponding t_p. In the classical backpropagation algorithm, the weights are changed according to the gradient descent direction of an error surface E.

Backpropagation

For backpropagation there are two updates performed, for the weights and the deltas. Lets begin with the weight update.

We are looking to compute $\frac{\partial E}{\partial w^l_{m',n'}}$ which can be interpreted as the measurement of how the change in a single pixel $w_{m',n'}$ in the weight kernel affects the loss function E.



During forward propagation, the convolution operation ensures that the yellow pixel $w_{m',n'}$ in the weight kernel makes a contribution in all the products (between each element of the weight kernel and the input feature map element it overlaps). This means that pixel $w_{m',n'}$ will eventually affect all the elements in the output feature map.

Convolution between the input feature map of dimension $H \times W$ and the weight kernel of dimension $k_1 \times k_2$ produces an output feature map of size $(H - k_1 + 1)$ by $(W - k_2 + 1)$. The gradient component for the individual weights can be obtained by applying the chain rule in the following way:

$$\begin{aligned}\frac{\partial E}{\partial w^l_{m',n'}} &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2}\frac{\partial E}{\partial x^l_{i,j}}\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \\ &= \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2}\delta^l_{i,j}\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} \end{aligned} \tag{10}$$

In Eq. 10, $x^l_{i,j}$ is equivalent to $\sum_m\sum_n w^l_{m,n}o^{l-1}_{i+m,j+n}+b^l$ and expanding this part of the equation gives us:

$$\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} = \frac{\partial}{\partial w^l_{m',n'}}\left(\sum_m\sum_n w^l_{m,n}o^{l-1}_{i+m,j+n}+b^l\right) \tag{11}$$

Further expanding the summations in Eq. 11 and taking the partial derivatives for all the components results in zero values for all except the components where $m=m'$ and $n=n'$ in $w^l_{m,n}o^{l-1}_{i+m,j+n}$ as follows:

$$\begin{aligned}\frac{\partial x^l_{i,j}}{\partial w^l_{m',n'}} &= \frac{\partial}{\partial w^l_{m',n'}}\left(w^l_{0,0}o^{l-1}_{i+0,j+0}+\cdots+w^l_{m',n'}o^{l-1}_{i+m',j+n'}+\cdots+b^l\right) \\ &= \frac{\partial}{\partial w^l_{m',n'}}\left(w^l_{m',n'}o^{l-1}_{i+m',j+n'}\right) \\ &= o^{l-1}_{i+m',j+n'} \end{aligned} \tag{12}$$

Substituting Eq. 12 in Eq. 10 gives us the following results:

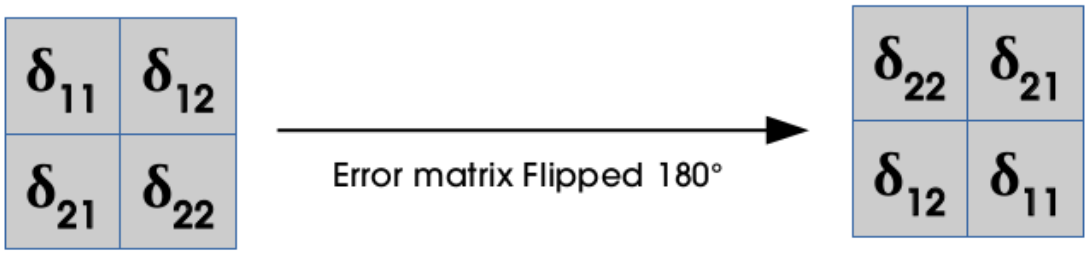
$$\frac{\partial E}{\partial w^l_{m',n'}} = \sum_{i=0}^{H-k_1}\sum_{j=0}^{W-k_2}\delta^l_{i,j}o^{l-1}_{i+m',j+n'} \tag{13}$$

$$= \mathrm{rot}_{180^\circ}\left\{\delta^l_{i,j}\right\} * o^{l-1}_{m',n'} \tag{14}$$

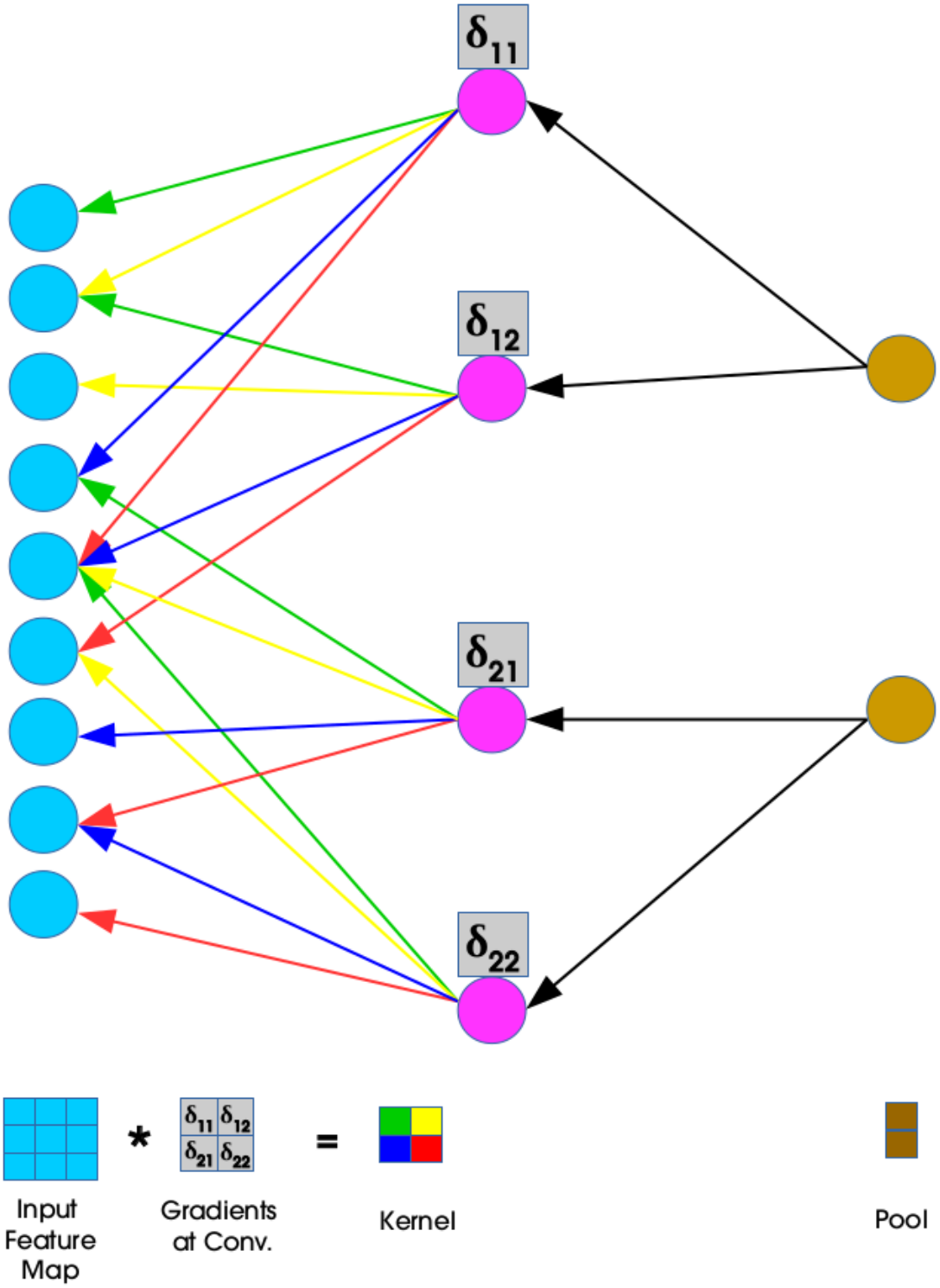
The dual summation in Eq. 13 is as a result of weight sharing in the network (same weight kernel is slid over all of the input feature map during convolution). The summations represents a collection of all the gradients $\delta^l_{i,j}$ coming from all the outputs in layer l.

Obtaining gradients w.r.t to the filter maps, we have a cross-correlation which is transformed to a convolution by "flipping" the delta matrix $\delta^l_{i,j}$ (horizontally and vertically) the same way we flipped the filters during the forward propagation.

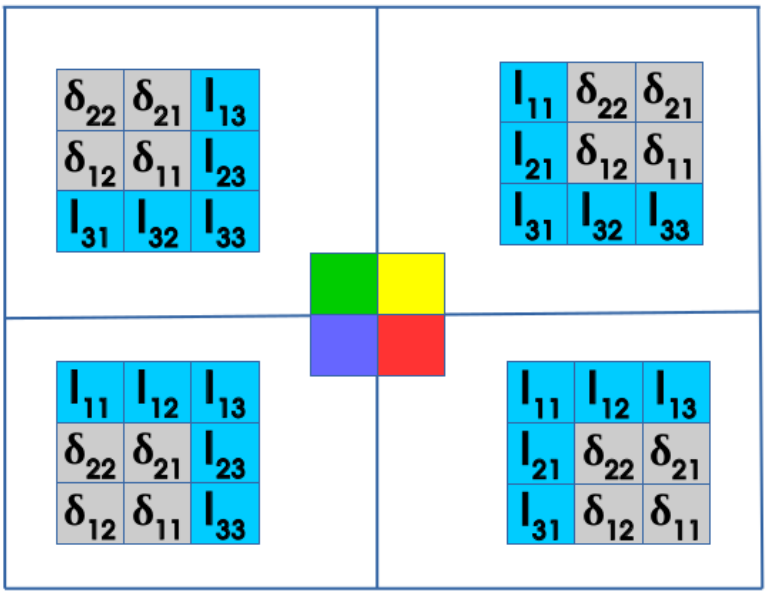
An illustration of the flipped delta matrix is shown below:



The diagram below shows gradients $(\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22})$ generated during backpropagation:



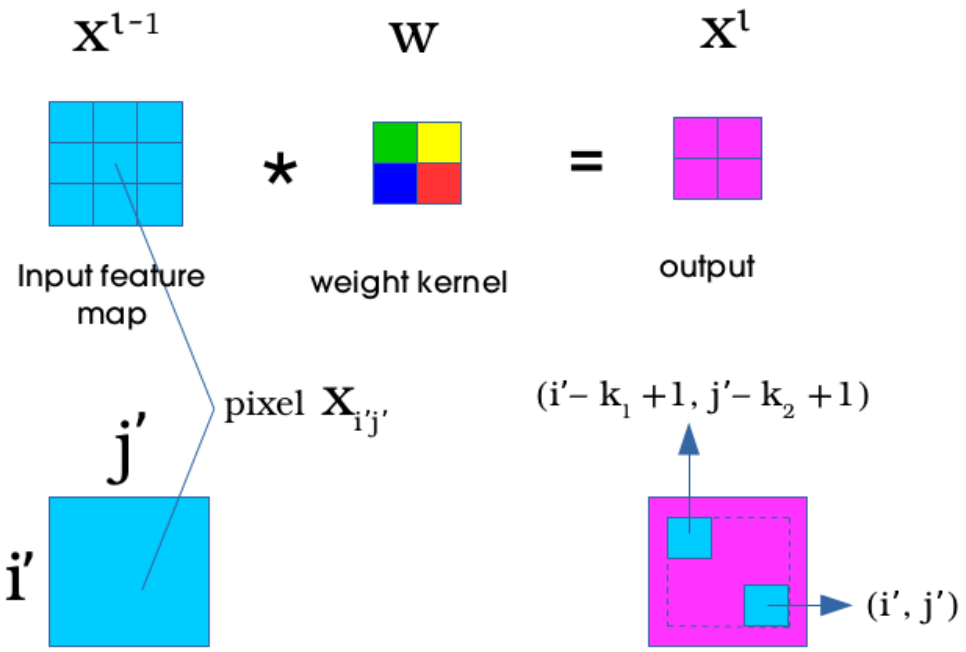
The convolution operation used to obtain the new set of weights as is shown below:



During the reconstruction process, the deltas $(\delta_{11}, \delta_{12}, \delta_{21}, \delta_{22})$ are used. These deltas are provided by an equation of the form:

$$\delta_{i,j}^{l} = \frac{\partial E}{\partial x_{i,j}^{l}} \tag{15}$$

At this point we are looking to compute $\frac{\partial E}{\partial x_{i',j'}^{l}}$ which can be interpreted as the measurement of how the change in a single pixel $x_{i',j'}$ in the input feature map affects the loss function E.



From the diagram above, we can see that region in the output affected by pixel $x_{i',j'}$ from the input is the region in the output bounded by the dashed lines where the top left corner pixel is given by $(i'-k_1+1, j'-k_2+1)$ and the bottom right corner pixel is given by (i', j').

Using chain rule and introducing sums give us the following equation:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{i,j \in Q} \frac{\partial E}{\partial x^{l+1}_Q} \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}} \\ &= \sum_{i,j \in Q} \delta^{l+1}_Q \frac{\partial x^{l+1}_Q}{\partial x^l_{i',j'}} \end{aligned} \tag{16}$$

Q in the summation above represents the output region bounded by dashed lines and is composed of pixels in the output that are affected by the single pixel $x_{i',j'}$ in the input feature map. A more formal way of representing Eq. 16 is:

$$\begin{aligned}\frac{\partial E}{\partial x^l_{i',j'}} &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \frac{\partial E}{\partial x^{l+1}_{i'-m,j'-n}} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \\ &= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} \frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} \end{aligned} \tag{17}$$

In the region Q, the height ranges from $i'-0$ to $i'-(k_1-1)$ and width $j'-0$ to $j'-(k_2-1)$. These two can simply be represented by $i'-m$ and $j'-n$ in the summation since the iterators m and n exists in the following similar ranges from $0 \leq m \leq k_1-1$ and $0 \leq n \leq k_2-1$.

In Eq. 17, $x^{l+1}_{i'-m,j'-n}$ is equivalent to $\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^l_{i'-m+m',j'-n+n'} + b^{l+1}$ and expanding this part of the equation gives us:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^l_{i',j'}} &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} o^l_{i'-m+m',j'-n+n'} + b^{l+1} \right) \\ &= \frac{\partial}{\partial x^l_{i',j'}} \left(\sum_{m'} \sum_{n'} w^{l+1}_{m',n'} f\left(x^l_{i'-m+m',j'-n+n'} \right) + b^{l+1} \right) \end{aligned} \tag{18}$$

Further expanding the summation in Eq. 17 and taking the partial derivatives for all the components results in zero values for all except the components where $m'=m$ and $n'=n$ so that $f\left(x^l_{i'-m+m',j'-n+n'}\right)$ becomes $f\left(x^l_{i',j'}\right)$ and $w^{l+1}_{m',n'}$ becomes $w^{l+1}_{m,n}$ in the relevant part of the expanded summation as follows:

$$\begin{aligned}\frac{\partial x^{l+1}_{i'-m,j'-n}}{\partial x^{l}_{i',j'}} &= \frac{\partial}{\partial x^{l}_{i',j'}}\left(w^{l+1}_{m',n'} f\left(x^{l}_{0-m+m',0-n+n'}\right) + \cdots + w^{l+1}_{m,n} f\left(x^{l}_{i',j'}\right) + \cdots + \right. \\ &= \frac{\partial}{\partial x^{l}_{i',j'}}\left(w^{l+1}_{m,n} f\left(x^{l}_{i',j'}\right)\right) \\ &= w^{l+1}_{m,n} \frac{\partial}{\partial x^{l}_{i',j'}}\left(f\left(x^{l}_{i',j'}\right)\right) \\ &= w^{l+1}_{m,n} f'\left(x^{l}_{i',j'}\right)\end{aligned}$$

Substituting Eq. 19 in Eq. 17 gives us the following results:

$$\frac{\partial E}{\partial x^{l}_{i',j'}} = \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} w^{l+1}_{m,n} f'\left(x^{l}_{i',j'}\right) \tag{20}$$

For backpropagation, we make use of the flipped kernel and as a result we will now have a convolution that is expressed as a cross-correlation with a flipped kernel:

$$\begin{aligned}\frac{\partial E}{\partial x^{l}_{i',j'}} &= \sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'-m,j'-n} w^{l+1}_{m,n} f'\left(x^{l}_{i',j'}\right) \\ &= \mathrm{rot}_{180^\circ}\left\{\sum_{m=0}^{k_1-1}\sum_{n=0}^{k_2-1} \delta^{l+1}_{i'+m,j'+n} w^{l+1}_{m,n}\right\} f'\left(x^{l}_{i',j'}\right) \qquad (21) \\ &= \delta^{l+1}_{i',j'} * \mathrm{rot}_{180^\circ}\left\{w^{l+1}_{m,n}\right\} f'\left(x^{l}_{i',j'}\right) \qquad (22)\end{aligned}$$

Pooling Layer

The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. No learning takes place on the pooling layers [2].

Pooling units are obtained using functions like max-pooling, average pooling and even L2-norm pooling. At the pooling layer, forward propagation results in an $N \times N$ pooling block being reduced to a single value - value of the “winning unit”. Backpropagation of the pooling layer then computes the error which is acquired by this single value “winning unit”.

To keep track of the “winning unit” its index noted during the forward pass and used for gradient routing during backpropagation. Gradient routing is done in the following ways:

- **Max-pooling** - the error is just assigned to where it comes from - the “winning unit” because other units in the previous layer’s pooling blocks did not contribute to it hence all the other assigned values of zero

- **Average pooling** - the error is multiplied by $\frac{1}{N \times N}$ and assigned to the whole pooling block (all units get this same value).

Conclusion

Convolutional neural networks employ a weight sharing strategy that leads to a significant reduction in the number of parameters that have to be learned. The presence of larger receptive field sizes of neurons in successive convolutional layers coupled with the presence of pooling layers also lead to translation invariance. As we have observed the derivations of forward and backward propagations will differ depending on what layer we are propagating through.

References

1. Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." stat 1050 (2016): 23. [PDF]
2. LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W.,Jackel, L.D.: Backpropagation applied to handwritten zip code recognition. Neural computation 1(4), 541–551 (1989)
3. Wikipedia page on Convolutional neural network
4. Convolutional Neural Networks (LeNet) deeplearning.net
5. Convolutional Neural Networks UFLDL Tutorial

Related Posts

Formulating The ReLu 24 Aug 2016

Vanishing And Exploding Gradient Problems 21 May 2018

75 Comments

Join the discussion…

21 Share Best

toosyou
6 years ago

Nice article!
However I found it difficult to tell whether '∗' is the notation of cross-correlatio convolution.
Or maybe it's just my misunderstanding?
Thank you for the great work either way :)

13 0 Reply • Share ›

Osama Khafagy
5 years ago

Awesome article, but flipping the kernel is not necessary, actually it causes co Thanks.

3 0 Reply • Share ›

Diego
4 years ago

NICE ARTICLE.I have been implementing a CNN but when I do the backpropag convolutional operations I have different depths so I can't do the convolutiona example:
Input 3x32x32 6 filters 3x5x5 and get 6×28×28 in the feed forward. In the back need to do the convolutional operation with the Delta's of the 6x28x28 with the that have different depth.

1 0 Reply • Share ›

jefkine kafuna Mod → Diego 4 years ago

Thanks! It is hard to tell where your challenge is without looking at yo implementation strategy.

0 0 Reply • Share ›

Diego → jefkine kafuna 4 years ago

How can I change the depth of a layer (or kernel) to have th convolutional operation?

0 0 Reply • Share ›

PhilS

4 years ago edited

Great stuff. At the backprop step I was missing the loss gradient w.r.t. the bias E / d b^l). However, when changing (10) with the bias derivative d x^l / d b^l thi one essentially with the result of the partial derivative of 1, what should make equal to sum_i sum_j delta^l_{i,j} \cdot 1 ?

1 0 Reply • Share ›

Osama Khafagy

5 years ago

Is this really the new the new kernel values? or the values to be added to the o (with including the learning rate factor)

1 0 Reply • Share ›

Jason XU → Osama Khafagy 5 years ago

That should be the gradient, so yes you still need to multiply that witl rate, the update the kernel by subtracting that lr*gradient thing.

0 0 Reply • Share ›

Titouan Parcollet

6 years ago

Hey, very nice article ! I'm still confuse about a thing. Regarding the BP and yo when it comes to update the weights I can see that the update of W1 will be a Delta(D)4 with I1 + D3*I2 + D2*I4 + D1*I5, but this make no sense right ? I mea update W1 with respect to some relations that doesn't exists no ? D4 * I1 are r Thanks !

1 0 Reply • Share ›

Derek Stinson → Titouan Parcollet 5 years ago

Its because the weights were flipped before convolving in this examp wouldn't do it that way. if you flip for convolution you have to flip the when updating the weights, but don't have to flip the weights to do th the next layer. If you don't flip for convolution you don't have to flip th update the weights, but you do have to flip the weights when you do convolution to send the errors to the previous layer. Also, I found it tc visualize updating the weights by doing this dw1=d1*i1, dw2=d1*i2, dw4=d1*i4. Then move to the right and do the next 4 and so on.

0 0 Reply • Share ›

Randy Swagmonster → Derek Stinson 5 years ago

Honestly, there's no use in using actual convolution when u Cross-correlation with no weight flipping at all is sufficient.

cross-correlation is essentially the case when we flip all of
weights to turn convolution to cross correlation and doing
assuming we never use a convolution operation.

0 0 Reply • Share ›

Steve Weavers

6 months ago

Thank you. I've been looking for this information for a while.

0 0 Reply • Share ›

byung gun kim

9 months ago

Thank you, It is a nice explanation!

0 0 Reply • Share ›

Teemu Sarapisto

a year ago

I think there is an error. "Subsequently for a bank of filters we have ... and bias
bank is 3 dimensional, not 4 dimensional here, right? I mean the convolution k
shared between the channels so the C dimension you have there shouldn't be
then in equation 4 only three dimensions are looped over.

0 0 Reply • Share ›

Teemu Sarapisto ↗ Teemu Sarapisto a year ago

Or I should say the same kernels are used for every channel so there
a channel dimension for the "bank of filters"

0 0 Reply • Share ›

Oleksandr

2 years ago

Thank you very much

0 0 Reply • Share ›

John Pikoulis

3 years ago

I have a question about the end result. The definition of the (*) convolution op
indexed with indices (i,j) denoting the (i,j) element of the output matrix resultin
convolution. That is the case with equation 14 and 22. However the partial der
weight w^(l)_{m',n'} in equation 14 is expressed as a convolution between a flip
and a delta^(l). But deltas are expressed as convolutions themselves so basic
saying in order to compute the final partial derivative, you are convolving a nur
matrix. Please correct me if i am wrong

0 0 Reply • Share ›

Akram Askar

3 years ago

Hi there!

Thanks for the article! Probably the best Backprop of CNN articles I've found s

I found a typing mistake btw... You are using different x_i,j^l notation in the not
equation 6 and 7. The bias term must be a mistake at the 6th & 7th equation is

0 0 Reply • Share ›

Satyam Dev Bhardwaj

4 years ago

hllw sir,
my structure of cnn is like con1 -> relu -> max -> con2 -> relu -> max -> flatten -
>softmax output
the problem is i can calculate the error upto the sec kernels or i can say i calcu
of sec kernel but the problem is how to pass my con2 layer or how to calculat
con2 layer. currently i m tring to figure this problem by doing deconvolution be
and kernel for getting the errors at con2 layer to get into my first layer.
sir if u have any suggestion or advise that will be very helpfull
sorry for worst eng if u didn't recognize my problem
n i m a learner but no community to learn so ur help will be very much helpful

thanking u
satyam

0 0 Reply • Share ›

Ruthvik Vaila

4 years ago

I have a question, if suppose I am workin on "correlational neural network", me
flip the kernel to begin with in forward propagation then I don't have to flip it in
but I have to flip the kernel to backward propagate the errors irrespective of cc
convolutional networks?

0 0 Reply • Share ›

Alan

4 years ago edited

Hey, very nice article.

I got a little confusion about equation 20 and 22. To me, equation 20 seems lik
convolution operation and it is equivalent to a cross-correlation with a flipped
equation 22 should be rot{W} $\otimes$ δ (I ignored the upper and lower index).

Thank you for the great work!

0 0 Reply • Share ›

Cici Hofsäß ↗ Alan 4 years ago

I was confused about that too :/
Do you get some more informations if this is really a correlation inste

convolution or do we understand something wrong?

0 0 Reply • Share ›

Jere Kabi ↗ Cici Hofsäß 4 years ago

A framework such as PyTorch actually does cross-correlati flipping). So I think you shouldn't worry. Hope this helps.

0 0 Reply • Share ›

Chandler Supple

5 years ago

Thank you so much! Admittedly, I didn't fully understand the article until readir time through. If you consider what the equations are doing intuitively, it's all qu

0 0 Reply • Share ›

jefkine kafuna Mod ↗ Chandler Supple 5 years ago

Awesome!

0 0 Reply • Share ›

ale83

5 years ago

Nice article. May be a double summation is missing in the sentence just abov

0 0 Reply • Share ›

jefkine kafuna Mod ↗ ale83 5 years ago

Thanks! have updated the article.

0 0 Reply • Share ›

raveesh s

5 years ago

Bro what is this i and j ?

0 0 Reply • Share ›

Michael Hakin

5 years ago

Great article! Thank you

0 0 Reply • Share ›

jefkine kafuna Mod → Michael Hakin 5 years ago

Thanks. Glad it was helpful.

0 0 Reply • Share ›

Andrea Apicella

5 years ago

beautiful article! but it is true if stride 1. What happens when stride is greater t convolution with a rotation of W?

0 0 Reply • Share ›

Jason XU

5 years ago edited

Hi Jefkine,

I got a small question regarding back-prop through max-pooling. Suppose a m pools from from layer l (o^l) to layer l+1 (o^{l+1}). To propagate delta^{l+1} to c picking out those corresponding to the maximum, do I need to multiply with th the activation in layer l?

0 0 Reply • Share ›

Jason XU

5 years ago

In Eq 14, are you sure it is rotating the delta matrix, not the o matrix?

0 0 Reply • Share ›

Jason XU → Jason XU 5 years ago edited

Ok I got it. So what's really happening is just a "sliding-window-produ is cross-correlation does. But it's called CNN and everyone calls "con make it mathematically consistent, here it is using a convolution (*) k kernel rotated. So I found it helps to get the head around whenever y ROT(sth), just replace with "sth sliding-window-product-sum".

The entire flipping thing is really confusing. And I think there is such the article as well. For instance in Fig 2 it shows "Input" * kernel, so a but the lines indicate a cross-correlation. And in Fig 5, showing still th forward prop by * (convolution), but the kernel is flipped, so one of th wrong.

And in Fig 5, I think the input image should be labelled O^{l-1} rather t

0 0 Reply • Share ›

Kai Waddington

5 years ago

Are the values obtained from equation 13/14 the values that the filter weights is there another step required to calculate the change of the filter weights?

0 0 Reply • Share ›

Josue

5 years ago

I am still confuse about how you got the deltas and how you applied it. What v the second layer had 8 neurons instead of 4?

0 0 Reply • Share ›

General Useage

5 years ago

AMAZING THANK YOU

0 0 Reply • Share ›

Amit

6 years ago edited

In the Notation part w_m,n^l is the scalar and not vector. w^l is the vector.

0 0 Reply • Share ›

This comment was marked as spam.

jefkine kafuna Mod → Peter Heinz 6 years ago

Thanks for the correction!! have updated the article.

0 0 Reply • Share ›

MT

6 years ago

Minor correction: Definition of w(l) should be between layer l and layer (l-1)

0 0 Reply • Share ›

jefkine kafuna Mod → MT 6 years ago

Thanks!! Have made the correction

0 0 Reply • Share ›

Yaswanth Reddy

6 years ago

Can you please explain the transition from equation 10 to 12. Could not figure Here is a snapshot of my working

0 0 Reply • Share ›

jefkine kafuna Mod → Yaswanth Reddy 6 years ago

Eq. 10 is the convolution equation. Eq 11 is the same convolution bu
a cross-corellation with a rotated kernel. The kernel is the delta_{i',j'}

0 0 Reply • Share ›

Yaswanth Reddy → jefkine kafuna 6 years ago

Isn't it the other way round, thinking Equation 10 can be tho
cross correlation delta X O (-i,-j) and we are converting it to
in Equation 12 ?

Also I dont get the negative value of the Cross correlation i
Does this mean that in the expansion of the Cross correlati
we are neglecting the terms with negative indices or are we
sort of negative indexing notation just like the one used in F

0 0 Reply • Share ›

jefkine kafuna Mod → Yaswanth Reddy 6 years ago

You might have to look at the definition of cross-c
its relation to convolution.

0 0 Reply • Share ›

Avata This comment was deleted.

jefkine kafuna Mod → Guest 6 years ago

Thanks for this, have updated the article

0 0 Reply • Share ›

Avata This comment was deleted.

jefkine kafuna Mod → Guest 6 years ago

The kind of convolution used for backprop is valid
where no zero-padding is done and kernel is restri
traverse only within the image. The output size: i.e
size w by w and kernel k by k we have an output (w
k + 1)

0 0 Reply • Share ›