Neural Networks and Deep Learning

Backprop, Convolution

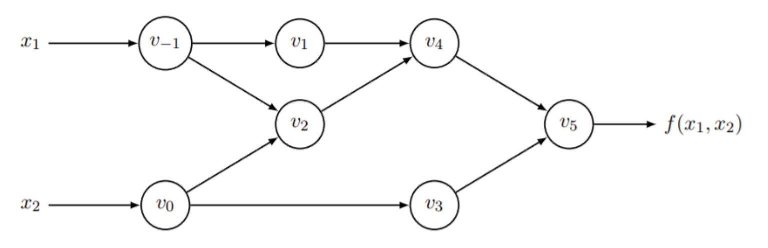
Adam Bloniarz

Department of Computer Science
adam.bloniarz@colorado.edu

February 4, 2020

Example

$$f(x_1, x_2) = ln(x_1) + x_1 * x_2 - sin(x2)$$



Computational graph for $f(x_1, x_2)$, Figure taken from Automatic differentiation in machine learning: A survey

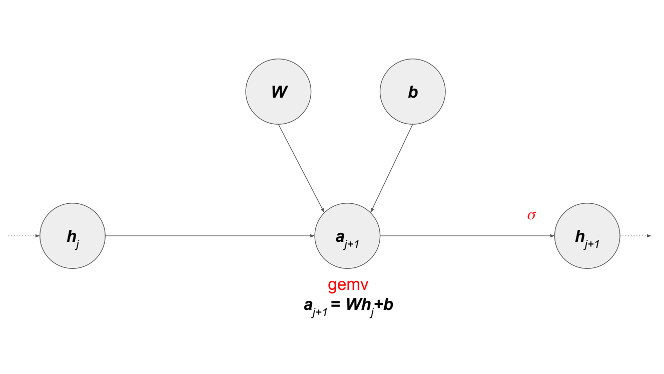
$$v_{-1} = x_1 \qquad v_0 = x_2$$

$$v_1 = ln(v_{-1}) \qquad v_2 = v_{-1}v_0$$

$$v_3 = sin(v_0) \qquad v_4 = v_1 + v_2$$

$$v_5 = v_4 - v_3 \qquad y = v_5$$

One layer of a feed-forward network.

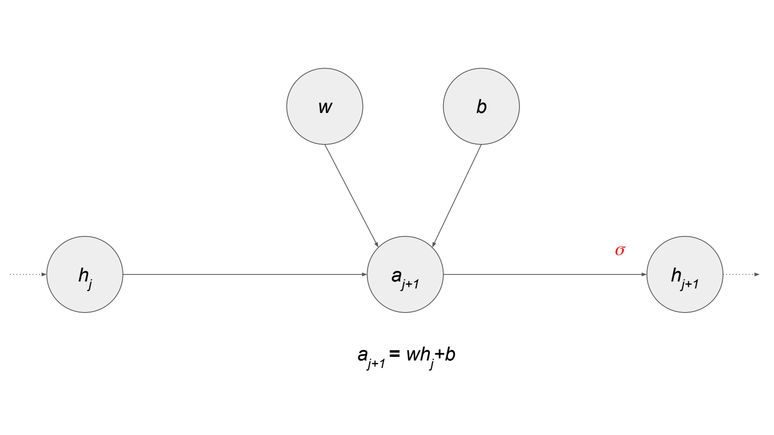


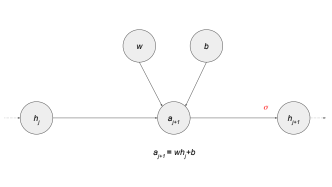
$\mathbf{h}_j \in \mathbb{R}^{n_j}$ $\qquad$ $\mathbf{h}_{j+1}, \mathbf{a}_{j+1} \in \mathbb{R}^{n_{j+1}}$

$\mathbf{W}$ is an $n_{j+1} \times n_j$ matrix.

$\mathbf{b} \in \mathbb{R}^{n_{j+1}}$

Scalar version





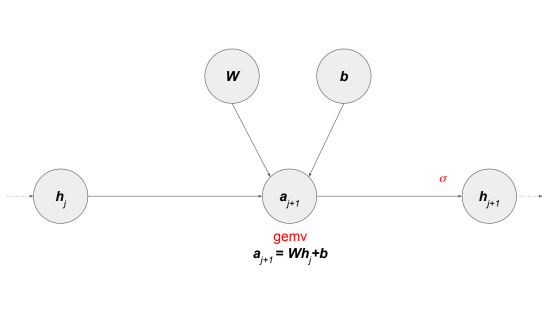
$$\underbrace{\frac{da_{j+1}}{dw}\frac{dL}{da_{j+1}}}_{\bar{w}=h_j\bar{a}_{j+1}}$$

$$\underbrace{\frac{da_{j+1}}{dh_j}\frac{dL}{da_{j+1}}}_{\bar{h}_j=w\bar{a}_{j+1}}$$

$$\underbrace{\frac{dh_{j+1}}{da_{j+1}}\frac{dL}{dh_{j+1}}}_{\bar{a}_{j+1}=\sigma'(a_j+1)\bar{h}_{j+1}}$$

$$\underbrace{\frac{dL}{dh_{j+1}}}_{\bar{h}_{j+1}}$$

$$\underbrace{\frac{da_{j+1}}{db}\frac{dL}{da_{j+1}}}_{\bar{b}=\bar{a}_{j+1}}$$

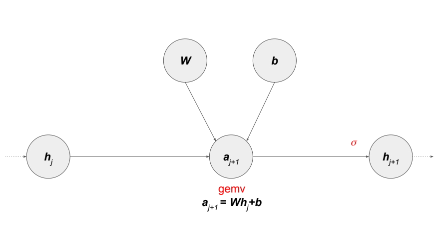


Input: $\nabla_{h_{j+1}} L$

$$h_{j+1} = \boldsymbol{\sigma}(a_j + 1)$$

Therefore,

$$\nabla_{a_{j+1}} L = \underbrace{\mathbf{J}_{\boldsymbol{\sigma}}(\mathbf{a}_{j+1})}_{\text{Jacobian of activation}} \nabla_{h_{j+1}} L$$



Input: $\nabla_{a_{j+1}} L$

$$\mathbf{a}_{j+1} = \mathbf{W}\mathbf{h}_j + \mathbf{b}$$

Therefore,

$$\nabla_{\mathbf{W}} L = \underbrace{\left(\nabla_{\mathbf{a}_{j+1}} L\right) \cdot \mathbf{h}_j^T}_{\text{outer product}}$$

$$\nabla_{\mathbf{b}} L = \nabla_{\mathbf{a}_{j+1}} L$$

$$\nabla_{\mathbf{h}_j} L = \mathbf{W}^T \left(\nabla_{\mathbf{a}_{j+1}} L\right)$$

Gradient of sof(arg)max activation function

$$\mathbf{p} = \boldsymbol{\sigma}(\mathbf{a})$$

$$\boldsymbol{\sigma} : \begin{bmatrix} a_1 \\ \vdots \\ a_p \end{bmatrix} \to \frac{1}{\sum_j e^{a_j}} \begin{bmatrix} e^{a_1} \\ \vdots \\ e^{a_p} \end{bmatrix}$$

What is the Jacobian $J_{\boldsymbol{\sigma}}(\boldsymbol{a})$?

What is $\frac{\partial p_i}{\partial a_i}$?

For $i = j$

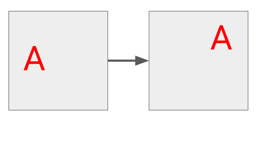
$$\frac{\partial p_i}{\partial a_i} = \frac{e^{a_i}\sum_k e^{a_k} - e^{2a_i}}{\left(\sum_k e^{a_k}\right)^2} = \left[\frac{e^{a_i}}{\sum_k e^{a_k}}\right]\left[\frac{\sum_k e^{a_k} - e^{a_i}}{\sum_k e^{a_k}}\right] = p_i\left(1 - p_i\right)$$

For $i \neq j$

$$\frac{\partial p_i}{\partial a_j} = -\frac{e^{a_i}}{\left(\sum_k e^{a_k}\right)^2} e^{a_j} = -p_i p_j$$

Why convolution?

Pixel representation is not stable under translation.



There is a weight associated to every pixel in the input domain.

There is no guarantee that the representation of ‘A’ on the left is the same as the representation of ‘A’ on the right.

The network must learn position invariance.

[MNIST colab]

Convolutional neural nets (CNNs) use the convolution operation.

Discrete convolution: each pixel is transformed into a fixed linear combination of its neighborhood.

The linear combination is known as a *filter*.

original image

0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

blur filter

result

original image

	0	1	0	
	1	-4	1	
	0	1	0	

edge detect filter

result

Source:https://docs.gimp.org/2.6/en/plug-in-convmatrix.html

Discrete Convolution

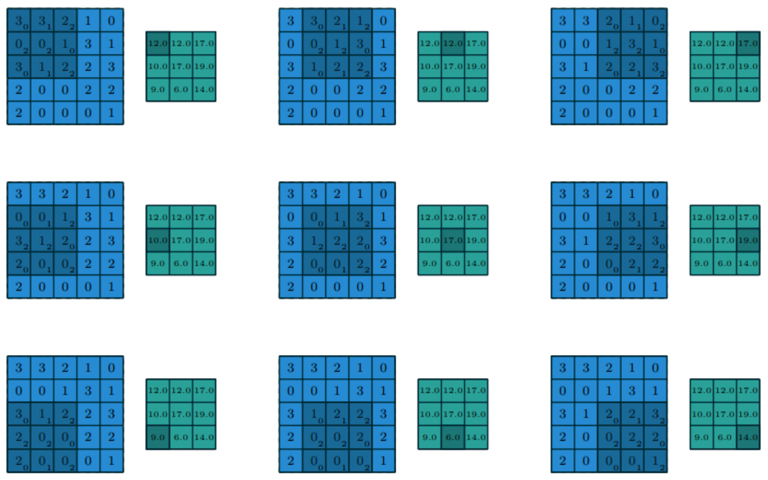


Image source: A guide to convolution arithmetic for deep learning.
blue: input feature map, *shaded blue*: kernel, *green*: output feature map.

Operator notation

Single channel

$$S(i,j) = (I * K)(i,j) = \int \int I(i-m, j-n)K(m,n)\,dm\,dn$$

Padding

What to do at the boundary?

- Zero padding is concatenating zeros around the border for convenience.
- The nice feature of zero padding is that it will allow us to control the spatial size of the output volumes.
- Types
  - » Padding schemes: valid, same
  - » Padding Values
    - Zero
    - Reflection
    - Replication
    - Constant

‘valid’ padding

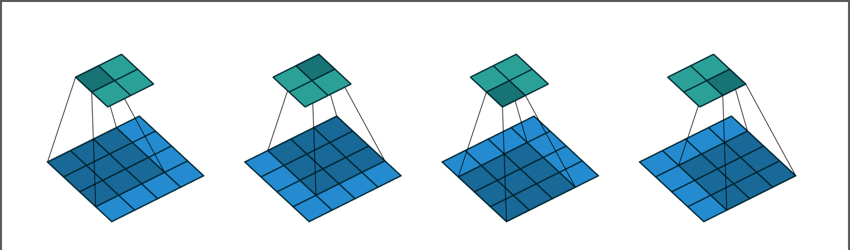


Image source: A guide to convolution arithmetic for deep learning.

For square kernel of size w, output shrinks by $w - 1$ in each dimension.

‘same’ padding

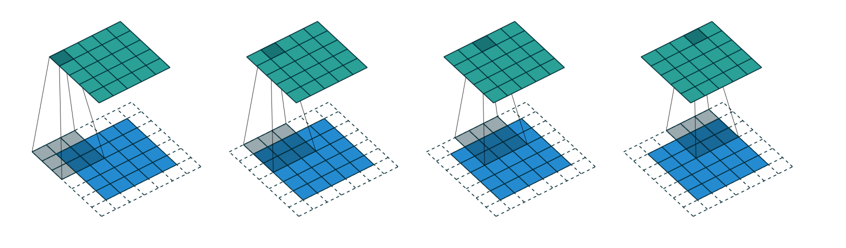


Image source: A guide to convolution arithmetic for deep learning.

Ensures output is the same height/width as input

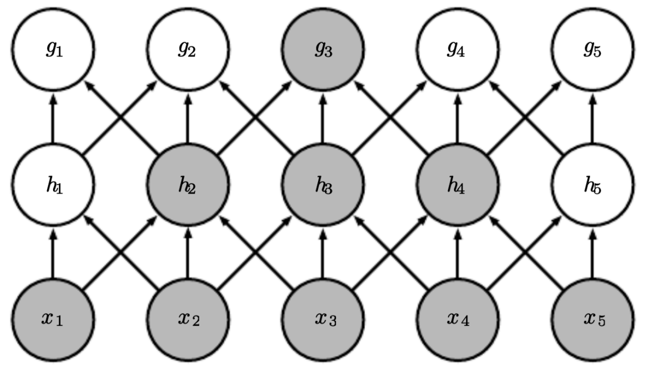
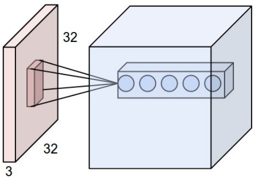


Image source: *Deep learning*, figure 9.4.

Stacking convolution layers increases the *receptive field* size of a unit.

Multiple channels



$$S(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \int\int \mathbf{I}(i-m, j-n)^T \mathbf{K}(m,n)\, dm dn$$

Allows for combinations of features across channels.

See also http://cs231n.github.io/convolutional-networks/

The number of convolutional filters in a layer is called the *depth*.
A 3x3 convolutional layer with:

Input: (height, width, depth_j)
Output: (height, width, depth_{j+1})

Requires depth_{j+1} filters, each of which is size $(3, 3, \text{depth}_j)$

Convolutional layer does not provide translation invariance, but rather *equivariance*.

Convolution operator commutes with translation operator:

Let $g_{u,v}$ be a shift operator:

$$(g_{u,v} \circ I)(i,j) = I(i-u, j-v)$$

Then

$$(g_{u,v} \circ I) * K = g_{u,v} \circ (I * K)$$