Math Foundations of ML – DL building blocks

Discrete Convolutions

1D discrete filter $g := \{g_k\}_{k=-M}^{M}$. Filter size $2M+1$.

Let $f = \{f_j\}_{j=-\infty}^{\infty}$. The discrete convolution is

$$f * g(k) := \sum_{j=-\infty}^{\infty} f_j g_{k-j} .$$

Examples:

1. Smooth convolution, low-pass filter $g = (g_{-1}, g_0, g_1) = \left[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right]$.
2. High-pass filter $g = (g_{-1}, g_0, g_1) = \left[-\frac{1}{4}, \frac{1}{2}, -\frac{1}{4}\right]$.

2D discrete filter $g := \{g_{k_1,k_2}\}_{k_1,k_2=-M}^{M}$. Filter size $(2M+1)^2$.

Let $f = \{f_j\}_{j\in\mathbb{Z}^2}$. The discrete convolution is

$$f * g(k) = f * g(k_1, k_2) := \sum_{j\in\mathbb{Z}^2} f_j g_{k-j} .$$

Example Smooth convolution, low-pass filter

$$g = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}$$

One can decompose a symmetric 2D filter through a tensor decomposition using a 1D filter $\tilde{g}$

$$f * g(k_1, k_2) = \left[\left[f(\cdot, k_2) * \tilde{g} \right](k_1, \cdot) * \tilde{g}\right](k_1, k_2)$$

Example $\tilde{g} = (\tilde{g}_{-1}, \tilde{g}_0, \tilde{g}_1) = \left[\frac{1}{4}, \frac{1}{2}, \frac{1}{4}\right]$

$$f * g(k_1,k_2) = \frac{1}{4}\left(\frac{1}{4} f_{k_1-1,k_2-1} + \frac{1}{2} f_{k_1-1,k_2} + \frac{1}{4} f_{k_1-1,k_2+1}\right)$$

i

$$+\frac{1}{2}\left(\frac{1}{4} f_{k_1,k_2-1} + \frac{1}{2} f_{k_1,k_2} + \frac{1}{4} f_{k_1,k_2+1}\right)$$

$$+\frac{1}{4}\left(\frac{1}{4} f_{k_1+1,k_2-1} + \frac{1}{2} f_{k_1+1,k_2} + \frac{1}{4} f_{k_1+1,k_2+1}\right)$$

What is this good for? In the Inception-B block for example, instead of using filters with dimension 7x7 in the x,y coordinates, there is a sequence of tensor 1D filters: 1x7, 7x1, 1x7, 7x1.

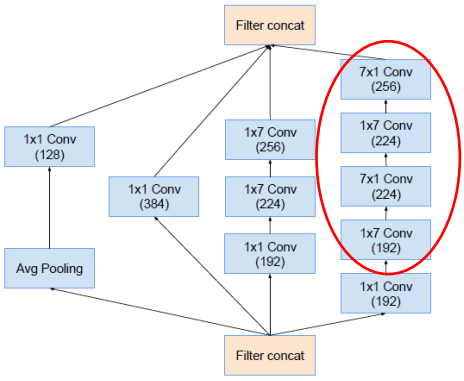
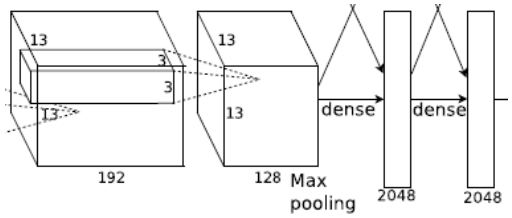


Figure 5. The schema for 17 × 17 grid modules of the pure Inception-v4 network. This is the Inception-B block of Figure 9.

In CNN for computer vision we typically need 3 dimensional convolutions

$$f * g(k) = f * g(k_1,k_2,k_3) := \sum_{j\in\mathbb{Z}^3} f_j g_{k-j} \,.$$

The filter is typically localized in the x,y direction, but not the z direction.

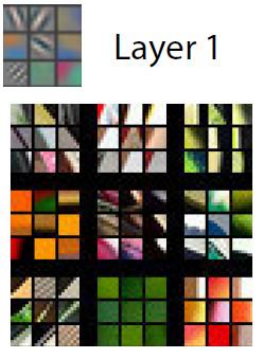


Trained convolutions of first layer – low level vision elements consisting of edge and color detection

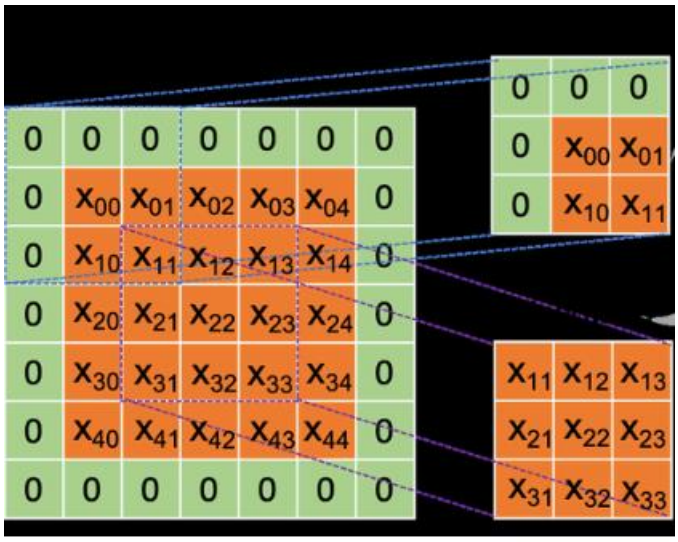
Visualizing and Understanding Convolutional Networks

Matthew D. Zeiler and Rob Fergus

Dept. of Computer Science,
New York University, USA
{zeiler,fergus}@cs.nyu.edu



Zero Padding for a 3x3 filter



Mirror/reflection padding

3	5	1
3	6	1
4	7	9

1	6	3	6	1	6	3
1	5	3	5	1	5	3
1	6	3	6	1	6	3
9	7	4	7	9	7	4
1	6	3	6	1	6	3

Fully connected (dense) layers

Definition Let $M \in M_{n\times m}$ be a nonsingular matrix and $b \in \mathbb{R}^m$. The associated affine transform $A: \mathbb{R}^n \to \mathbb{R}^m$, is defined by $Ax = Mx + b$, $\forall x \in \mathbb{R}^n$.

Fully connected layer – every input vector element potentially contributes to any output vector element $x_{out} = Ax_{in} = Mx_{in} + b$.

Remark A convolutional layer allows to significantly reduce the number of (unknown) weights, based on the assumption of locality, i.e. the output (neurons) have a localization property, they are associated with visual elements in a certain neighborhood (whose support grows with the depth of the layers). So, they should only be affected by inputs in a certain neighborhood. So regardless of the size of the feature maps, the number of weights of a convolution layer is determined by the number of features maps X the number of convolutions X the size of the convolution filters. This is not (!) the case with dense layers.

Non-linearities

A decomposition of affine transforms is an affine transform

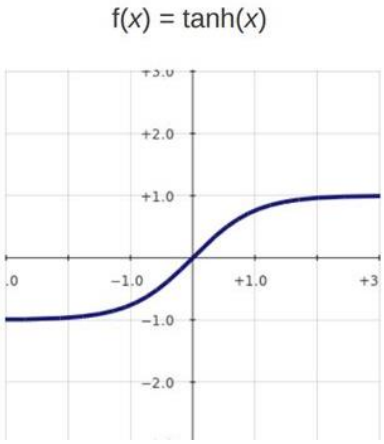
$$A_2\left(A_1x\right) = M_2\left(A_1x\right) + b_2 = M_2\left(M_1x + b_1\right) + b_2 = \left(M_2M_1\right)x + \left(M_2b_1 + b_2\right).$$

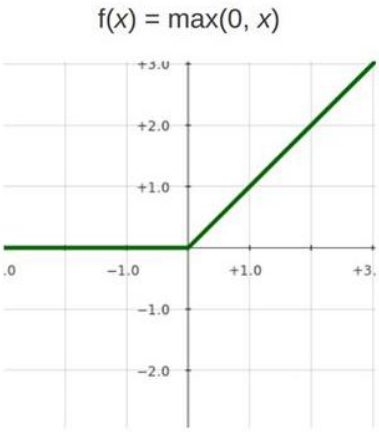
So, in a way, without any other functionality a convolutional neural network can be essentially collapsed into one affine transform.

Typical non-linearities

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{\frac{e^x - e^{-x}}{2}}{\frac{e^x + e^{-x}}{2}} = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

ReLU (Rectifier Linear Unit)

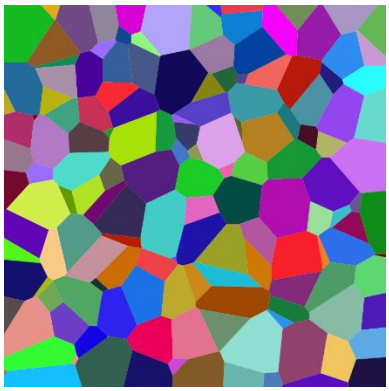




Application of the non-linearities Typically applied pointwise after the affine operations. So if $\sigma(y) := (\mathrm{ReLU}(y_1), \ldots, \mathrm{ReLU}(y_m))$, then we apply $\sigma(A(x)) = \sigma(Mx + b)$.

Non-linearities can be explained in two ways:

1. Neural Sciences approach - Simulation of neuron activation
2. Approximation-theoretical explanation of ReLu – The entire NN can be viewed as a piecewise linear approximation over the original feature space. Why? Over sub-domains of the original feature subspace, the NN collapses into one local affine transform.



Example Let $\Omega \subset \mathbb{R}^n$ be a domain. Then for $x \in \Omega$ and $\sigma(y) := (\mathrm{ReLU}(y_1), \ldots, \mathrm{ReLU}(y_m))$, the application $\sigma(A(x)) = \sigma(Mx + b)$, can split up Ω into 2^n sub-domains over which $\sigma \circ A$ is an affine transform.

Remark NN as a piecewise linear approximation? Why aren't we studying this theory? I've been aware of this 'perspective' for many years, however, I'm not confident that this is theoretical foundation that explains the success of DL. As we've discussed, the datasets in the original feature space have very low 'smoothness'/structure… so any approximation algorithm, even adaptive piecewise linear approximation, is problematic to analyze. The theoretical analysis of the 'unfolding' across layers and the improvement of the smoothness of the layer functions, is in fact more in line with the DL community's way of thinking about the transformation taking place across layers… from low-level vision features to high-level features that allow classification, etc.

Residual Blocks ([2016], cited 45,241 times)

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

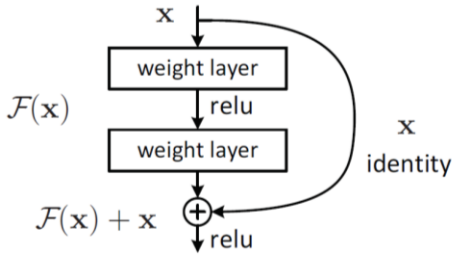


Figure 2. Residual learning: a building block.

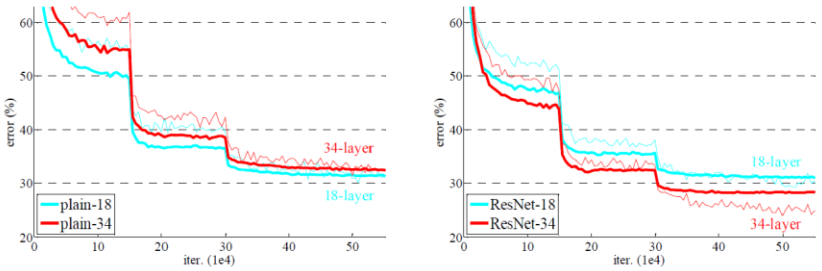
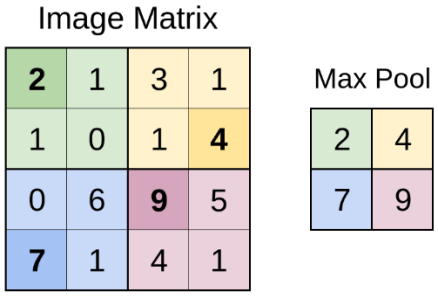


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

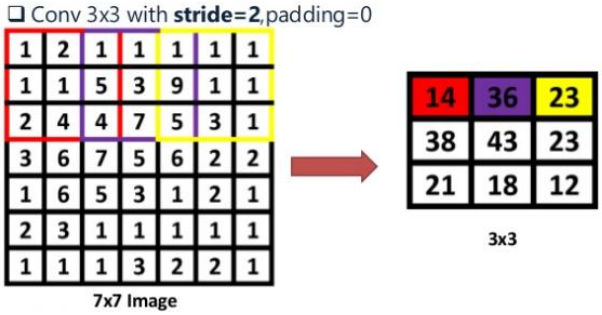
Pooling

Pooling – a tool for dimensionality reduction / further neuron activation. Typically applied after the non-linearities

Max-pooling



Strides



MNIST dataset of hand-written numbers 0-9. Each image is 28x28

Back propagation

Inference model $f(x) = f(x, w)$

Loss $L(w) = \sum_i \left(f(x_i, w) - y_i \right)^2$

Minimization via gradient descent (Newton's method). Start with initial guess w^0. At each step k compute for a certain subset $\Lambda^{(k)}$ of the training dataset

$$\begin{aligned} \frac{d}{dw} L \Big|_{w^{(k)}} &= 2 \sum_{i \in \Lambda^{(k)}} \left(f\left(x_i, w^{(k)}\right) - y_i \right) \frac{\partial f\left(x_i, w^{(k)}\right)}{\partial w} \\ &= 2 \left\langle f\left(x_i, w^{(k)}\right) - y_i, \frac{\partial f\left(x_i, w^{(k)}\right)}{\partial w} \right\rangle_{\Lambda^{(k)}} \end{aligned}$$

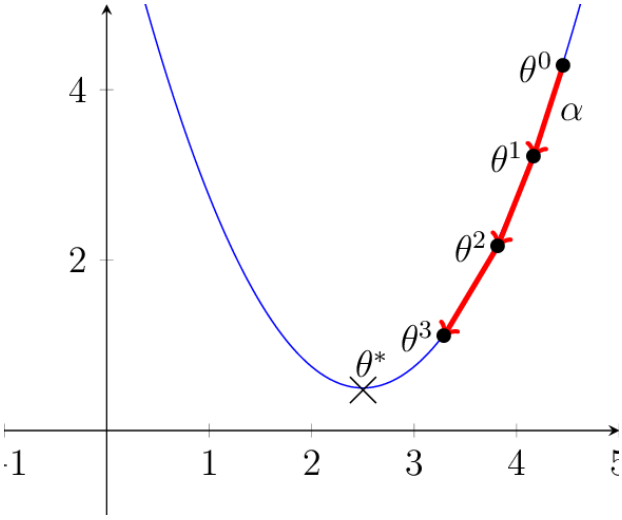
Now suppose inference is using a model $f(x) = f_2\left(f_1(x, w_1), w_2\right)$, $w_1, w_2 \in \mathbb{R}$, $w = (w_1, w_2)$,

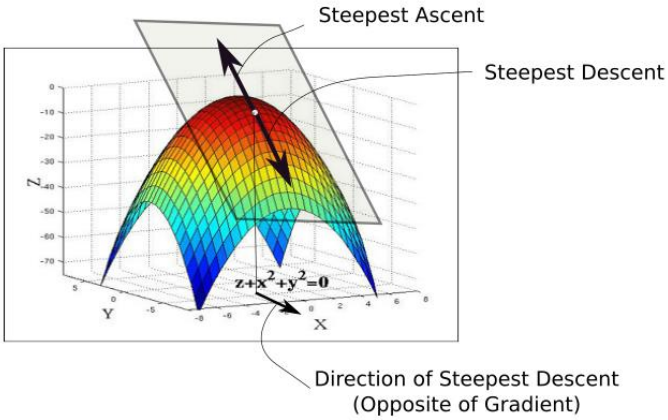
$$\begin{aligned} \frac{d}{dw_2} L \Big|_{w^{(k)}} &= 2 \sum_{i \in \Lambda^{(k)}} \left(f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right) - y_i \right) \frac{\partial f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right)}{\partial w_2} \\ &= 2 \left\langle f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right) - y_i, \frac{\partial f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right)}{\partial w_2} \right\rangle_{\Lambda^{(k)}} \end{aligned}$$

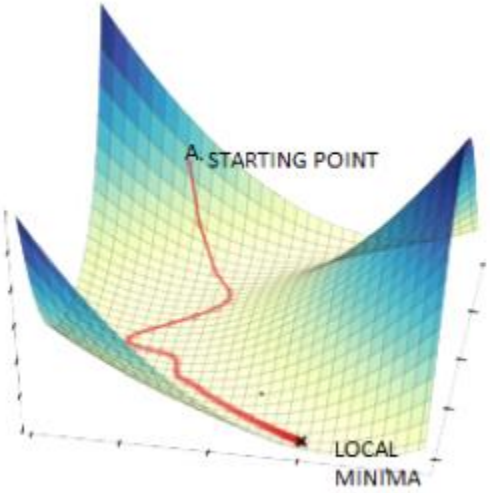
$$\frac{d}{dw_1} L \Big|_{w^{(k)}} = 2 \sum_{i \in \Lambda^{(k)}} \left(f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right) - y_i \right) \frac{\partial f_2\left(f_1\left(x_i, w_1^{(k)}\right), w_2^{(k)} \right)}{\partial x_1} \frac{\partial f_1\left(x_i, w_1^{(k)}\right)}{\partial w_1}$$

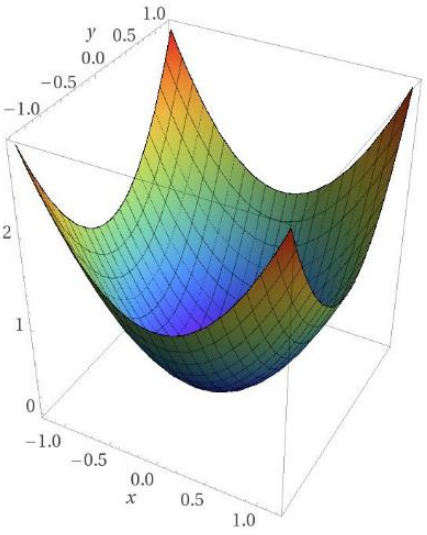
So the first layer passes to the second layer the information $\left\{ f_1\left(x_i, w_1^{(k)}\right) \right\}_{\Lambda^{(k)}}, \left\{ \frac{\partial f_1\left(x_i, w_1^{(k)}\right)}{\partial w_1} \right\}_{\Lambda^{(k)}}$.

Gradient descent

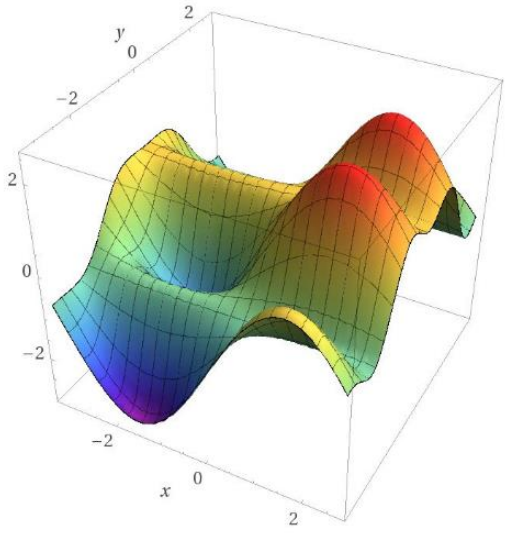








Computed by WolframAlpha



Computed by WolframAlpha