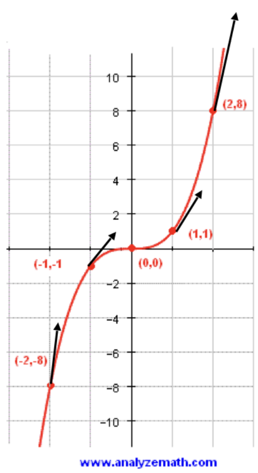
CAS CS 440
Lec 25

Supervised Learning VII – Neural Networks (cont.)

1. Direction of Gradient
    a. Lets say we have a (differentiable) function 'f' we want to minimize
    b. The derivative of 'f' is a direction
        i. Points to local maxima (including saddle points)
    c. We can use this to "descend!"
        i. Move in opposite direction of derivative!
    d. Basic algorithm (in 1-D):
        i.
        while not converged and $t < t_{max}$ do:
        compute $\frac{df}{dx}|_{x^{(t)}}$
        update $x^{(t+1)} = x^{(t)} - \eta \frac{df}{dx}|_{x^{(t)}}$
        $t = t + 1$
    e. 

2. Gradient Descent in Parameter Space
    a. Remember, we want to find $\boldsymbol{\theta}$ that optimizes a loss function
    b. So, we need to compute

$$\nabla_{\theta} L(\boldsymbol{D}, \boldsymbol{\theta}) = \left(\frac{\partial L}{\partial \theta_1}, \frac{\partial L}{\partial \theta_2}, \ldots, \frac{\partial L}{\partial \theta_n}\right)^T$$

    c. Basic algorithm (in parameter space)

while not converged and $t < t_{max}$ do:

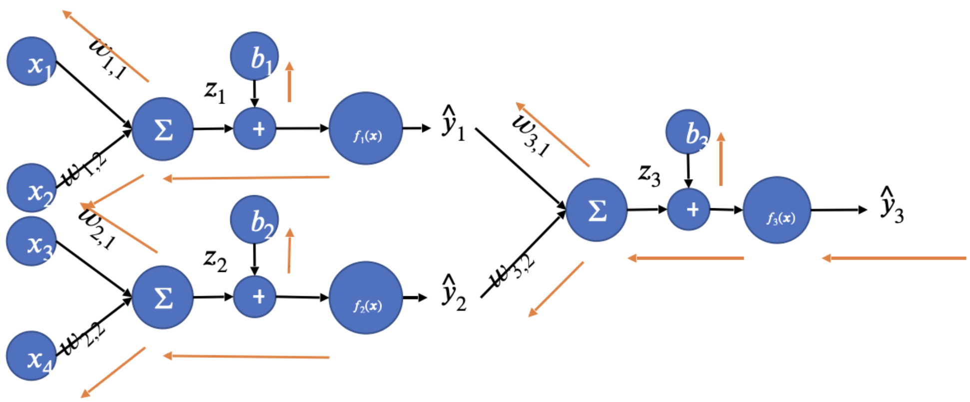
compute $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{D}, \boldsymbol{\theta})\Big|_{\boldsymbol{\theta}^{(t)}}$

update $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} L(\boldsymbol{D}, \boldsymbol{\theta})\,|_{x^{(t)}}$

$t = t + 1$

i.

3. Computing the Gradient



a.

b. Gradient "flows" backward (from output to input)

c. Lets do a small example, see if we notice any patterns

Look at all the redundant calculations!!!

$$\nabla_{\boldsymbol{\theta}} L = \begin{matrix} \frac{\partial L}{\partial b_3} \\ \frac{\partial L}{\partial w_{3,1}} \\ \frac{\partial L}{\partial w_{3,2}} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial w_{1,1}} \\ \frac{\partial L}{\partial w_{1,2}} \\ \cdots \end{matrix} = \begin{matrix} \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \\ \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial w_{3,1}} \\ \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial w_{3,2}} \\ \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \\ \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,1}} \\ \frac{\partial L}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3} \frac{\partial z_3}{\partial \hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1} \frac{\partial z_1}{\partial w_{1,2}} \\ \cdots \end{matrix}$$

i.

ii. Now, we cache the backpropagation values and use it again for later (to remove redundancy)

4. Backpropagation
    a. NN computation is always a DAG
    b. Cache error as a function of layer
        i. Remove redundant computation (from previous slide)

$$\delta_{\hat{y}_3} = \frac{\partial L}{\partial \hat{y}_3}$$

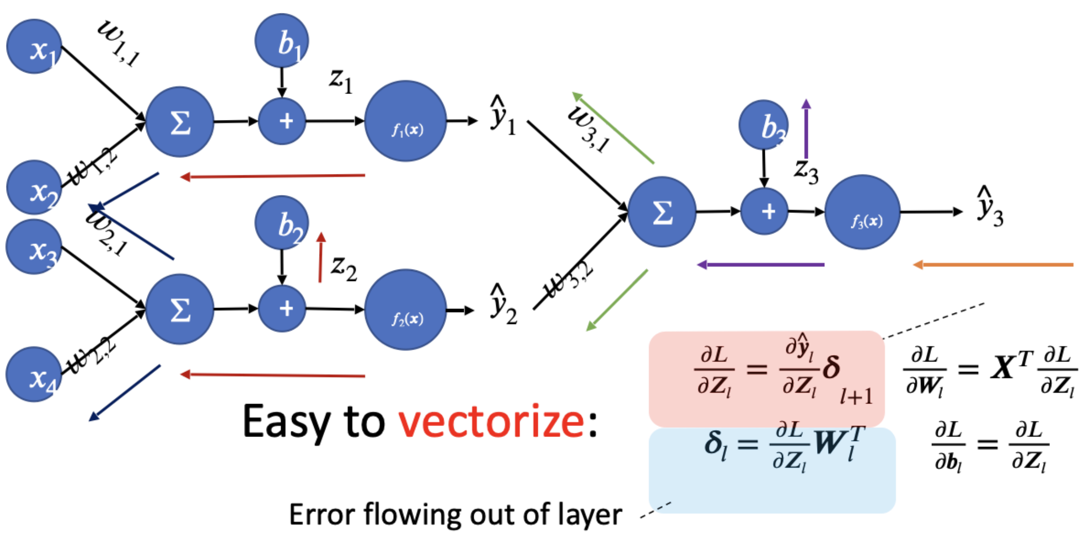
$$\delta z_3 = \delta_{\hat{y}_3} \frac{\partial \hat{y}_3}{\partial z_3}$$

$$\delta_{\hat{y}_1} = \delta z_3 \frac{\partial z_3}{\partial \hat{y}_1}$$

$$\delta z_1 = \delta_{\hat{y}_1} \frac{\partial \hat{y}_1}{\partial z_1}$$

$$\nabla_\theta L = \begin{array}{c} \frac{\partial L}{\partial b_3} \\ \frac{\partial L}{\partial w_{3,1}} \\ \frac{\partial L}{\partial w_{3,2}} \\ \frac{\partial L}{\partial b_1} \\ \frac{\partial L}{\partial w_{1,1}} \\ \frac{\partial L}{\partial w_{1,2}} \\ \cdots \end{array} = \begin{array}{l} \delta_{\hat{y}_3}\ \delta z_3 \\ \delta_{\hat{y}_3}\ \delta z_3\ \frac{\partial z_3}{\partial w_{3,1}} \\ \delta_{\hat{y}_3}\ \delta z_3\ \frac{\partial z_3}{\partial w_{3,2}} \\ \delta_{\hat{y}_3}\ \delta z_3\ \delta_{\hat{y}_1}\ \delta z_1 \\ \delta_{\hat{y}_3}\ \delta z_3\ \delta_{\hat{y}_1}\ \delta z_1\ \frac{\partial z_1}{\partial w_{1,1}} \\ \delta_{\hat{y}_3}\ \delta z_3\ \delta_{\hat{y}_1}\ \delta z_1\ \frac{\partial z_1}{\partial w_{1,2}} \\ \cdots \end{array}$$

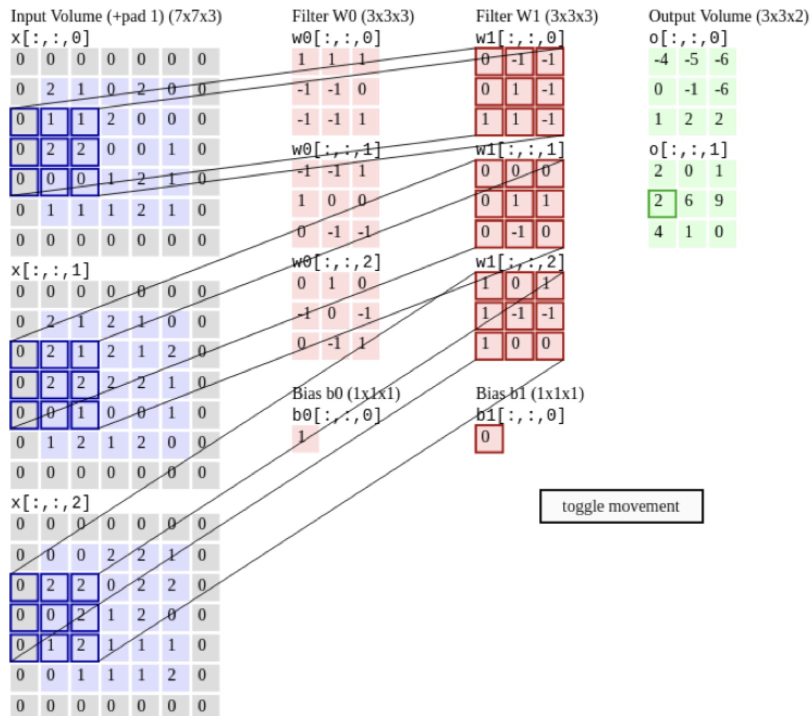
Error from activation func

    c. 

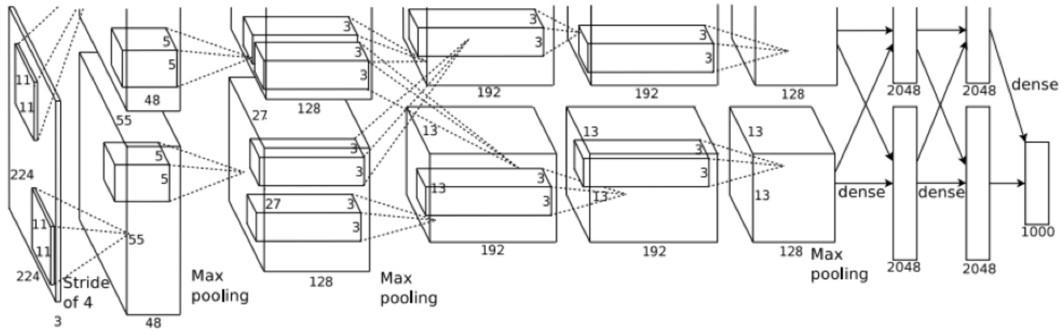
    d.
    e. Calculate once and reuse it (cache the matrix that has the values)
    f. Data inputs have to be vectors

5. Images and Your Eyes
    a. How do your eyes parse visual data?
    b. How do we localize different patterns in images?
        i. Answer: Convolution!

No statistical difference between eye response and convolution with gabor (wavelet) filters[1]



    c.
    d. High level: convolution "rubs" one function (called the filter or kernel) over a signal (image)
        i. Response signal is proportional to how similar geometry in signal is to filter
        ii. High response -> local geometry of signal is similar to geometry of filter!
    e. Can we learn the filters?
6. Convolutional NNs
    a. Implement convolution operator (over images) using learnable kernels
    b. Generate kernels (like gabor wavelets!)
    c. Successive layers → more abstract features



d.

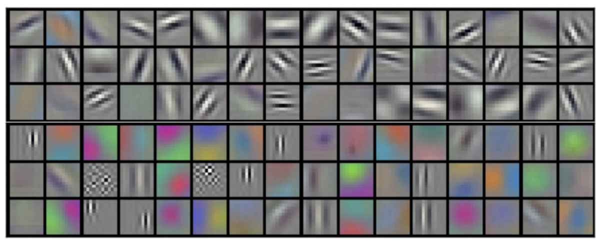


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

e.