

Spring School on
Deep Learning for Medical Imaging (DLMI2023)
Lyon, France | April 17, 2023

Basics of deep learning : Part II

Christian Desrosiers

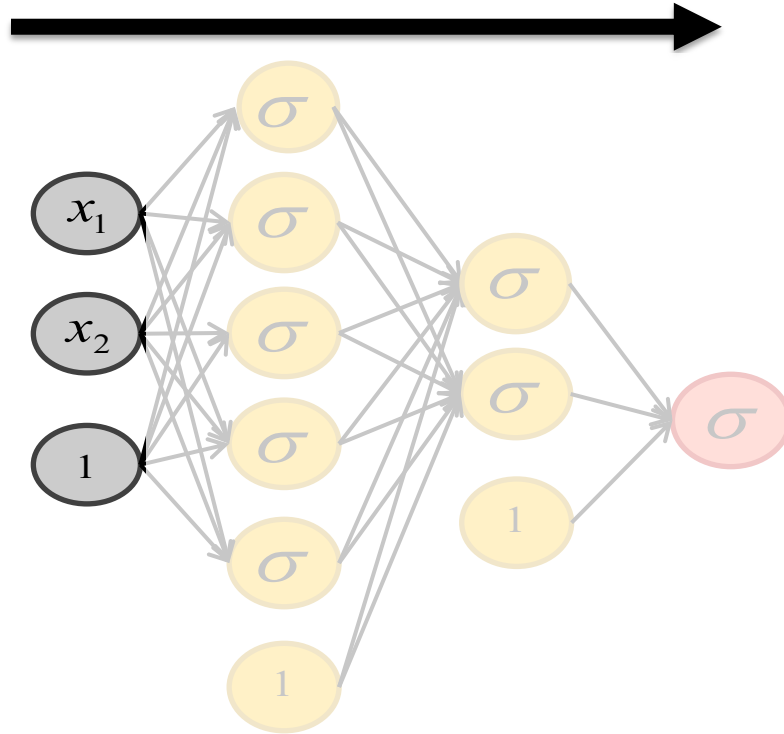
ÉTS Montreal, Canada



How to train the network?

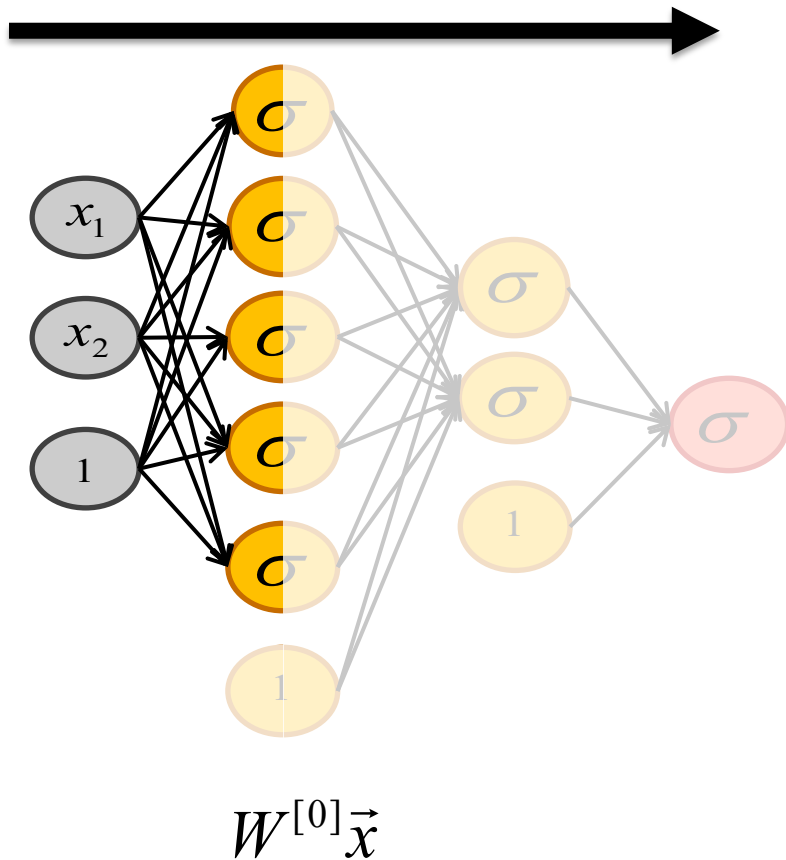


Forward pass

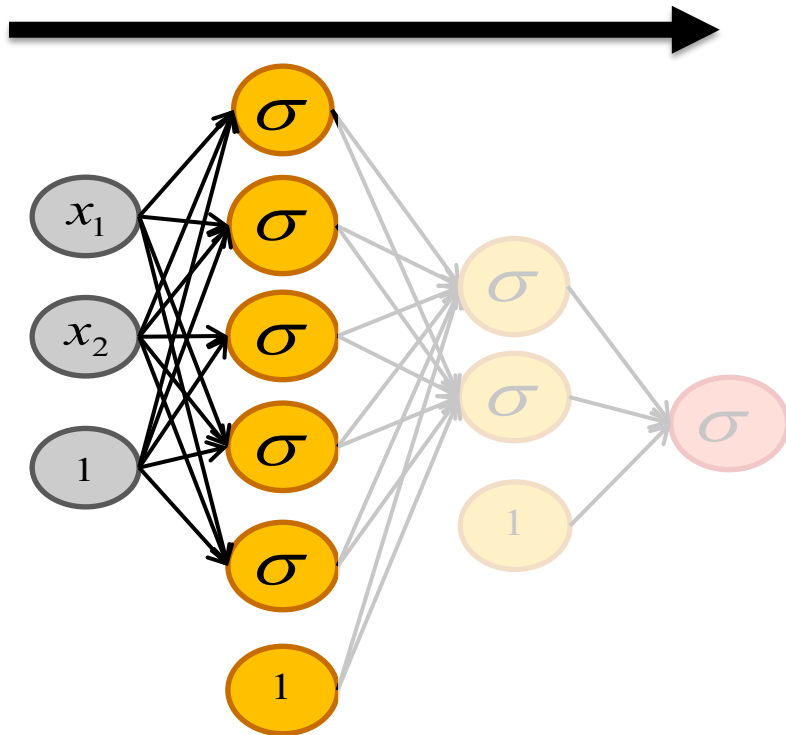


\vec{x}

Forward pass

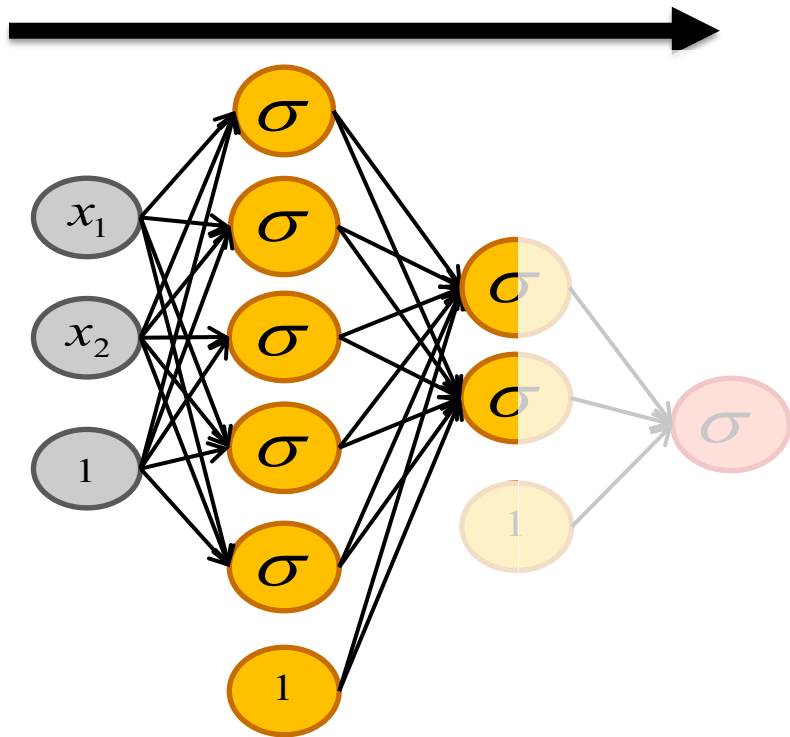


Forward pass



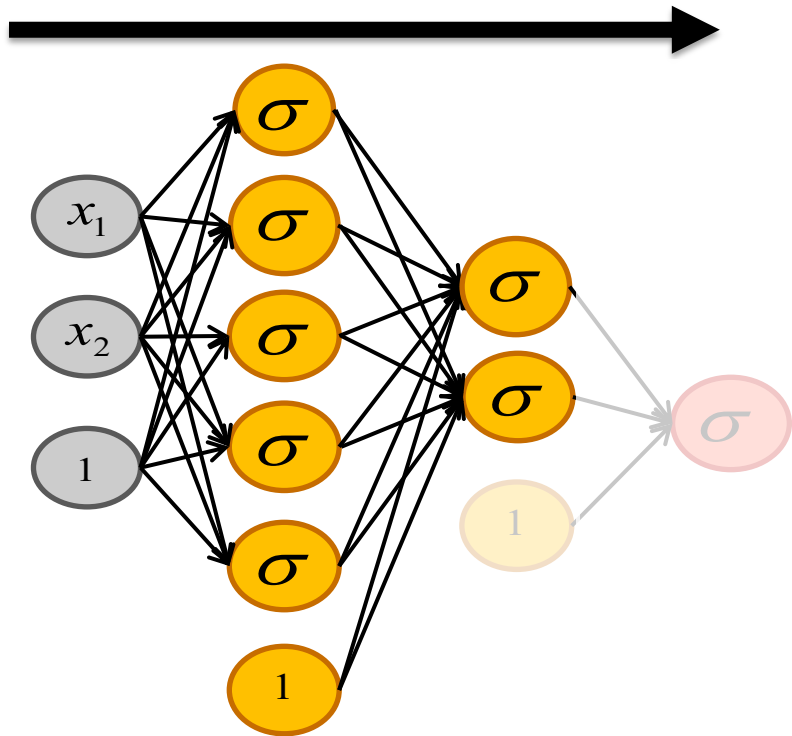
$$\sigma(W^{[0]}\vec{x})$$

Forward pass



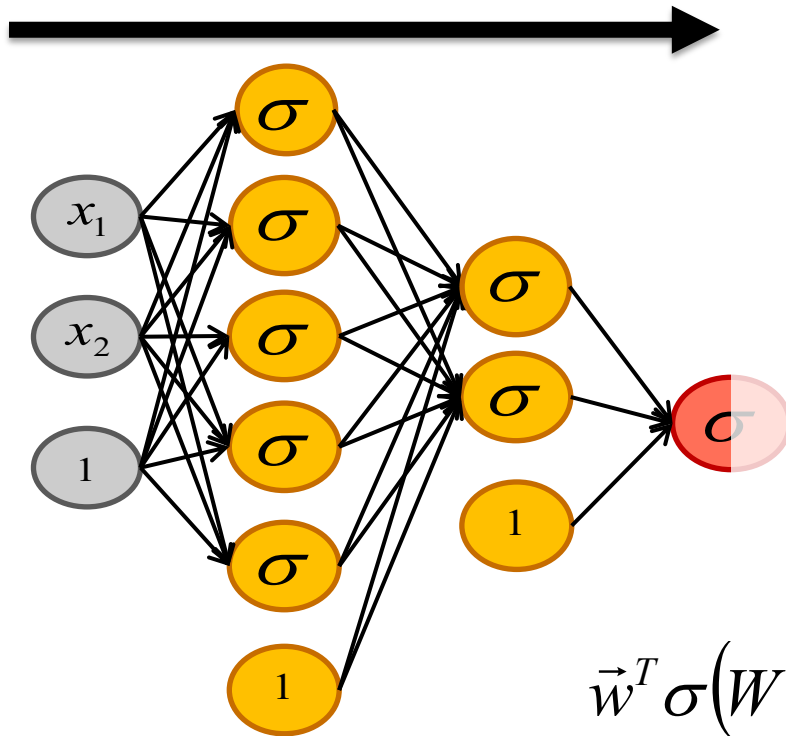
$$W^{[1]} \sigma(W^{[0]} \vec{x})$$

Forward pass



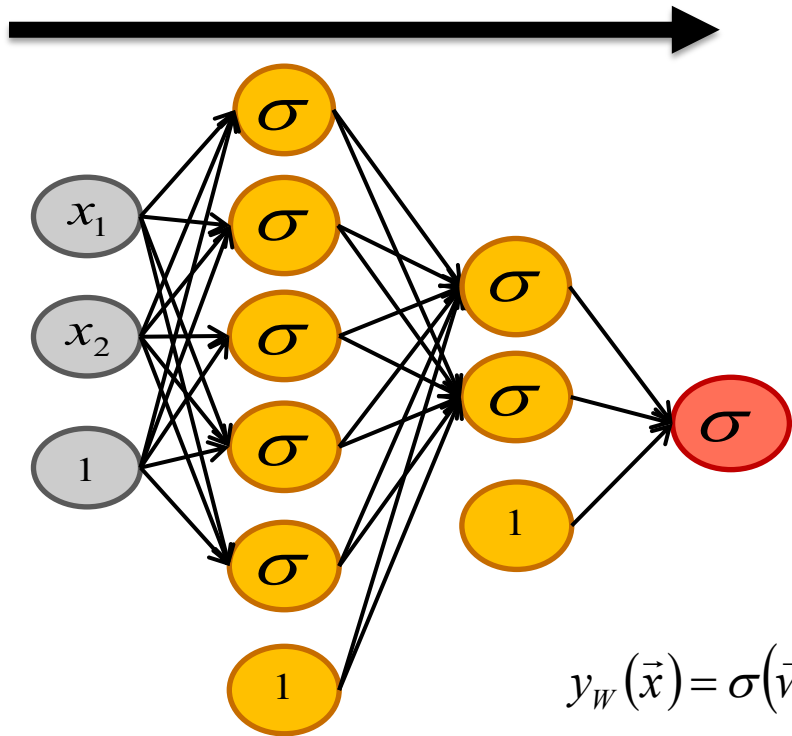
$$\sigma(W^{[1]}\sigma(W^{[0]}\vec{x}))$$

Forward pass



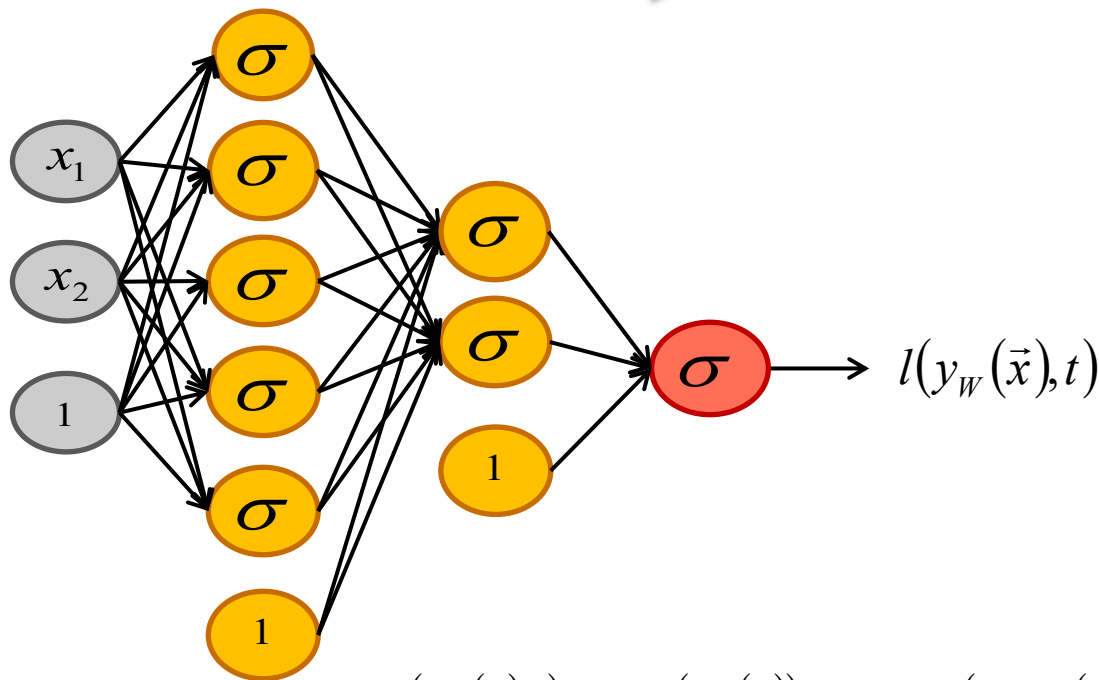
$$\vec{w}^T \sigma(W^{[1]} \sigma(W^{[0]} \vec{x}))$$

Forward pass



$$y_W(\vec{x}) = \sigma(\vec{w}^T \sigma(W^{[1]} \sigma(W^{[0]} \vec{x})))$$

Forward pass



$$l(y_w(\vec{x}), t) = -t \ln(y_w(\vec{x})) - (1-t) \ln(1 - y_w(\vec{x}))$$

How to optimize the network?

1) From

$$W = \arg \min_W = \sum_{n=1}^N l(y_W(\vec{x}_n), t_n) + \lambda R(W)$$

Choose a regularization function

$$R(W) = \|W\|_1 \text{ or } \|W\|_2^2$$

How to optimize the network?

2) Choose a loss $l(y_W(\vec{x}_n), t_n)$, for example

Hinge loss

Cross entropy



Do not forget to adjust the output layer with the loss you have chosen.

Cross entropy => Softmax

How to optimize the network?

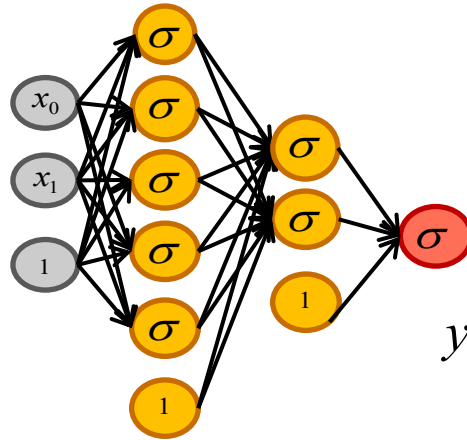
3) Compute the gradient of the loss with respect to each parameter

$$\frac{\partial \left(\sum_{n=1}^N l(y_W(\vec{x}_n), t_n) + \lambda R(W) \right)}{\partial W_{a,b}^{[c]}}$$

and update the parameters using gradient descent

$$W_{a,b}^{[c]} := W_{a,b}^{[c]} - \eta \frac{\partial \left(\sum_{n=1}^N l(y_W(\vec{x}_n), t_n) + \lambda R(W) \right)}{\partial W_{a,b}^{[c]}}$$

How to optimize the network?

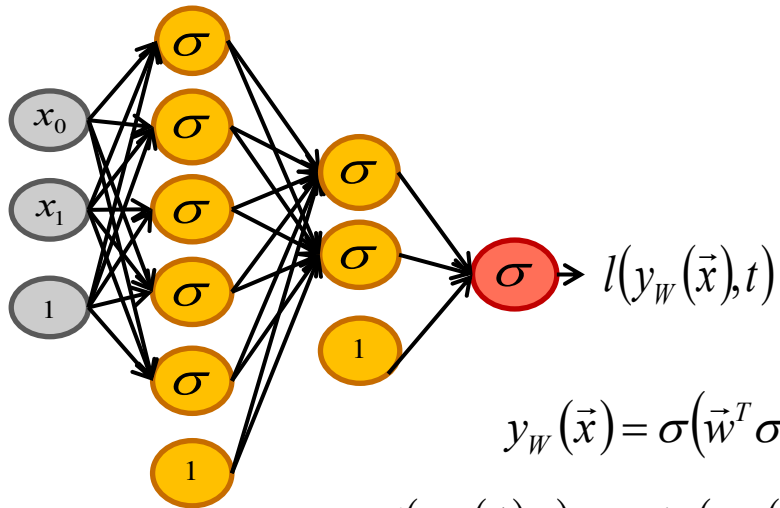


$$y_W(\vec{x}) = \sigma(\vec{w}^{[2]} \sigma(W^{[1]} \sigma(W^{[0]} \vec{x})))$$

$$\frac{\partial \left(\sum_{n=1}^N l(y_W(\vec{x}_n), t_n) + \lambda R(W) \right)}{\partial W_{a,b}^{[c]}}$$



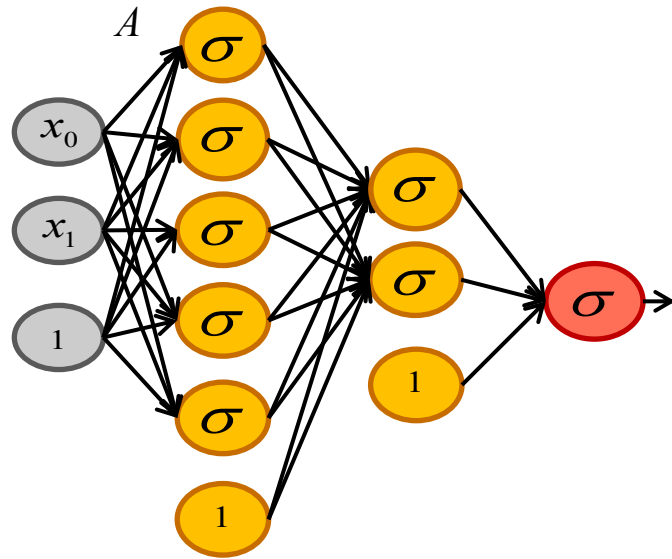
Backpropagation



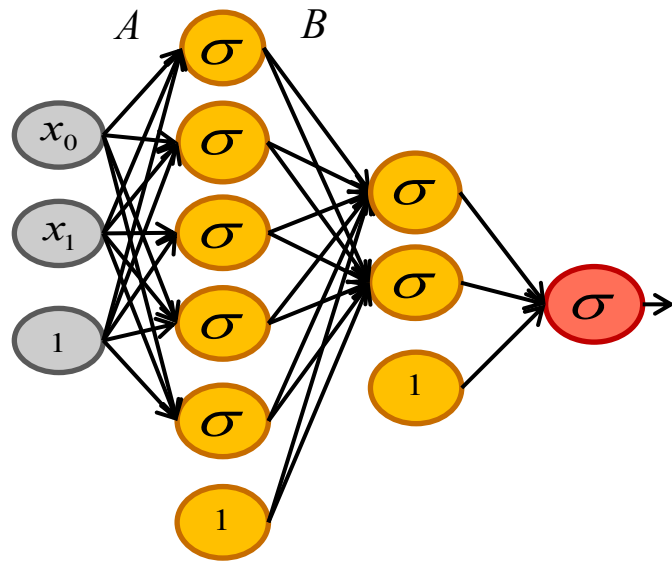
$$y_W(\vec{x}) = \sigma(\vec{w}^T \sigma(W^{[1]} \sigma(W^{[0]} \vec{x})))$$

$$l(y_W(\vec{x}), t) = -t \ln(y_W(\vec{x})) - (1-t) \ln(1 - y_W(\vec{x}))$$

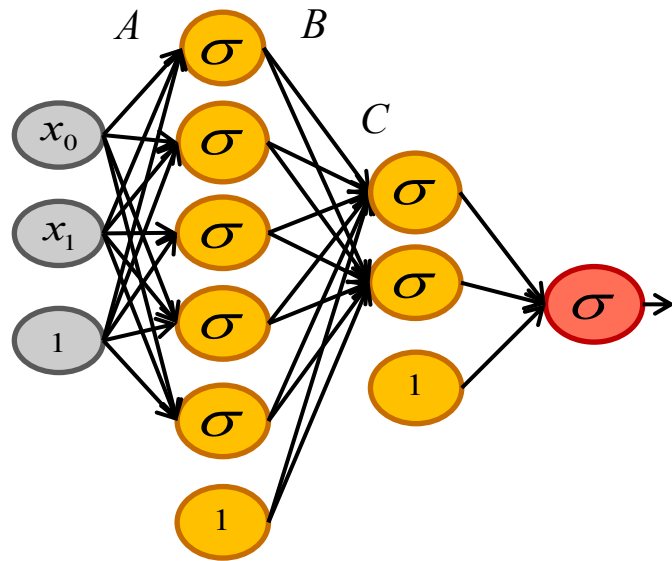
$$A = W^{[0]} \vec{x}$$



$$A = W^{[0]} \vec{x}$$
$$B = \sigma(A)$$



$$A = W^{[0]}\vec{x}$$
$$B = \sigma(A)$$
$$C = W^{[1]}B$$

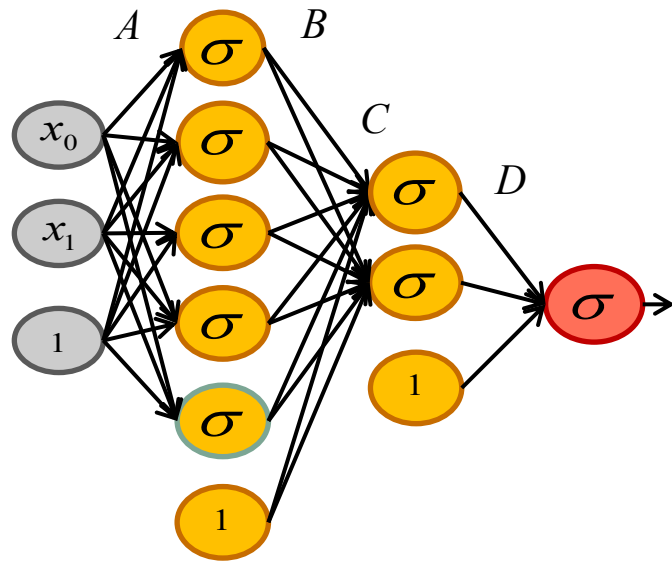


$$A = W^{[0]} \vec{x}$$

$$B = \sigma(A)$$

$$C = W^{[1]} B$$

$$D = \sigma(C)$$



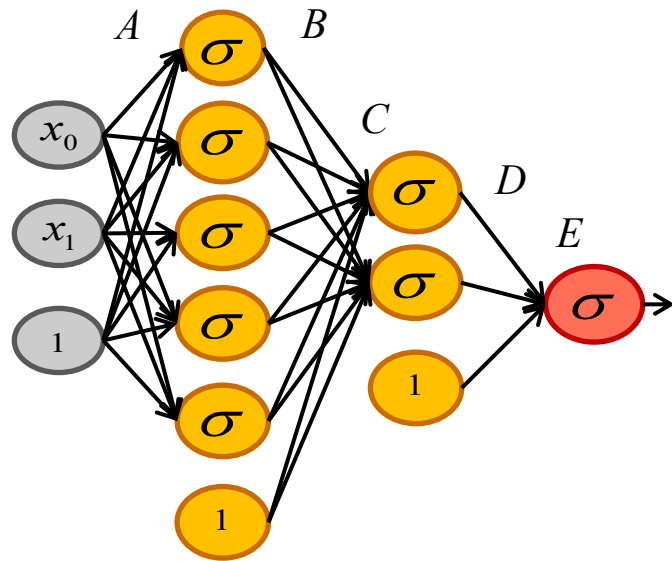
$$A = W^{[0]} \vec{x}$$

$$B = \sigma(A)$$

$$C = W^{[1]} B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$



$$A = W^{[0]}\vec{x}$$

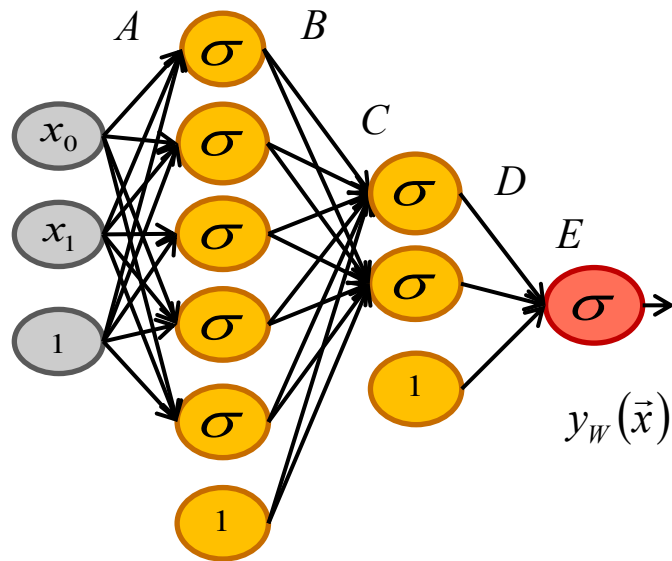
$$B = \sigma(A)$$

$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$



$$y_W(\vec{x}) = \sigma(\vec{w}^T \sigma(W^{[1]} \sigma(W^{[0]}\vec{x})))$$

$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

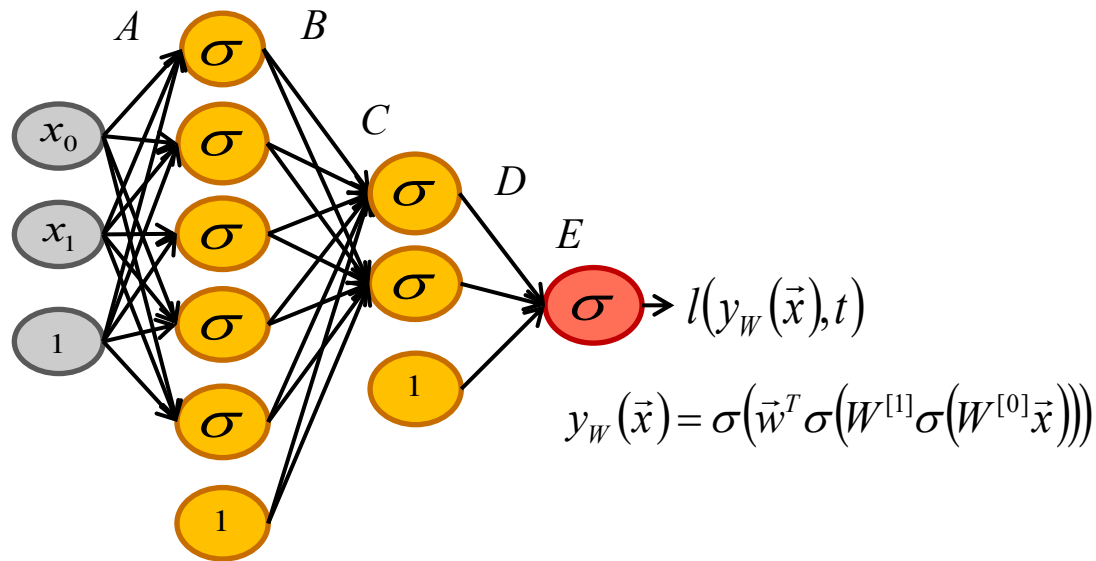
$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$

$$l(y_W(\vec{x}), t)$$



$$\frac{\partial \left(\sum_{n=1}^N l(y_W(\vec{x}_n), t_n) \right)}{\partial W^{[l]}} \quad ?$$

Chain rule



Chain rule recap

$$\left. \begin{array}{l} f(u) = u^2 \\ u(v) = 2v \\ v(x) = 1/x \end{array} \right\} \frac{\partial f}{\partial x} = ? \left\{ \begin{array}{l} \frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \times \frac{\partial u}{\partial v} \times \frac{\partial v}{\partial x} \\ = 2u \times 2 \times \left(-\frac{1}{x^2} \right) \end{array} \right.$$

$$A = W^{[0]}\vec{x}$$

$$B = \sigma(A)$$

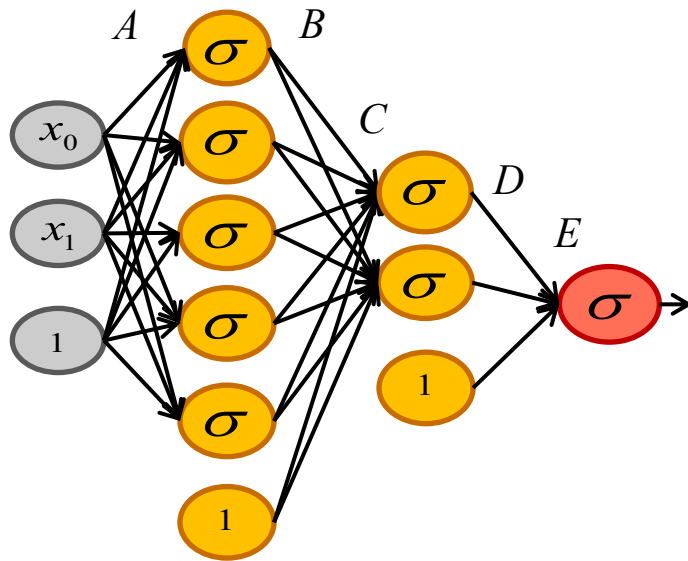
$$C = W^{[1]}B$$

$$D = \sigma(C)$$

$$E = \vec{w}^T D$$

$$y_W(\vec{x}) = \sigma(E)$$

$$l(y_W(\vec{x}), t)$$

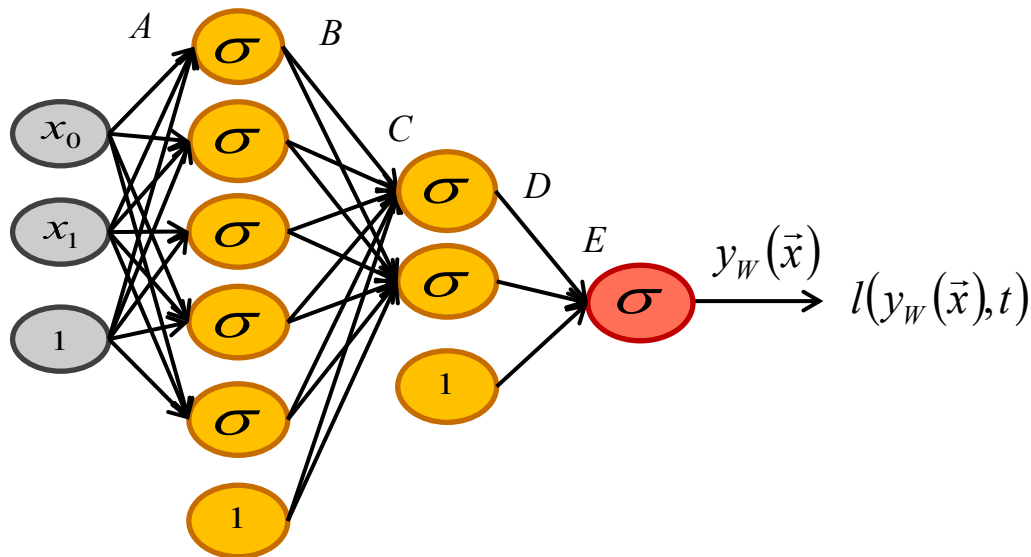


$$\frac{\partial(l(y_W(\vec{x}), t))}{\partial W^{[0]}} = \frac{\partial(l(y_W(\vec{x}), t))}{\partial y_W(\vec{x})} \frac{\partial(y_W(\vec{x}))}{\partial E} \frac{\partial(E)}{\partial D} \frac{\partial(D)}{\partial C} \frac{\partial(C)}{\partial B} \frac{\partial(B)}{\partial A} \frac{\partial(A)}{\partial W^{[0]}}$$

Back propagation



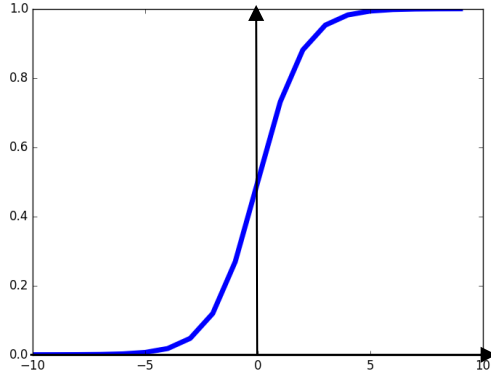
$$\frac{\partial(l(y_W(\vec{x}), t))}{\partial W^{[0]}} = \frac{\partial(l(y_W(\vec{x}), t))}{\partial y_W(\vec{x})} \frac{\partial(y_W(\vec{x}))}{\partial E} \frac{\partial(E)}{\partial D} \frac{\partial(D)}{\partial C} \frac{\partial(C)}{\partial B} \frac{\partial(B)}{\partial A} \frac{\partial(B)}{\partial W^{[0]}}$$



Activation functions

Activation functions

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

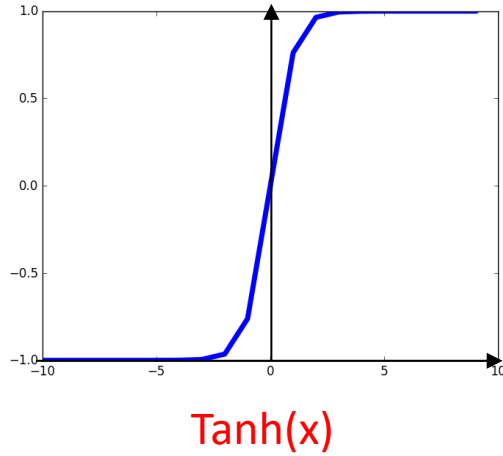


Sigmoid

3 Problems :

- Gradient saturates when input is large
- Not zero centered
- $\exp()$ is an expensive operation

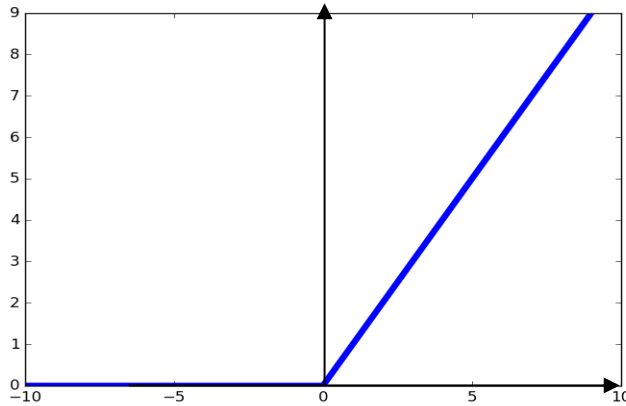
Activation functions



- Output is zero-centered 😊
- Small gradient when input is large 😞

Activation functions

$$\text{ReLU}(x) = \max(0, x)$$

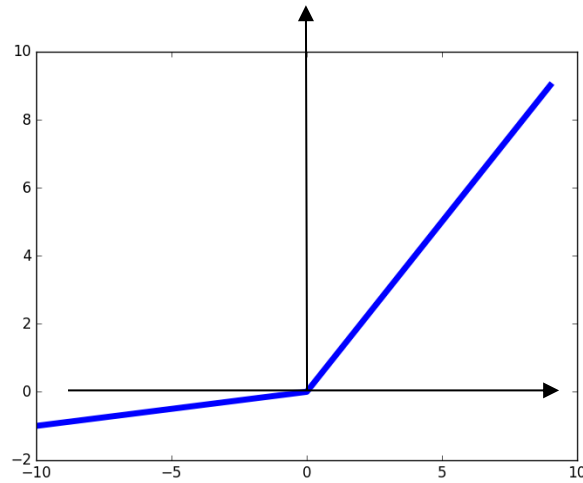


ReLU(x)
(Rectified Linear Unit)

- Large gradient for $x > 0$ 😊
- Super fast 😊
- Output non centered at zero 😞
- No gradient when $x < 0$? 😞

Activation functions

$$\text{LReLU}(x) = \max(0.01x, x)$$



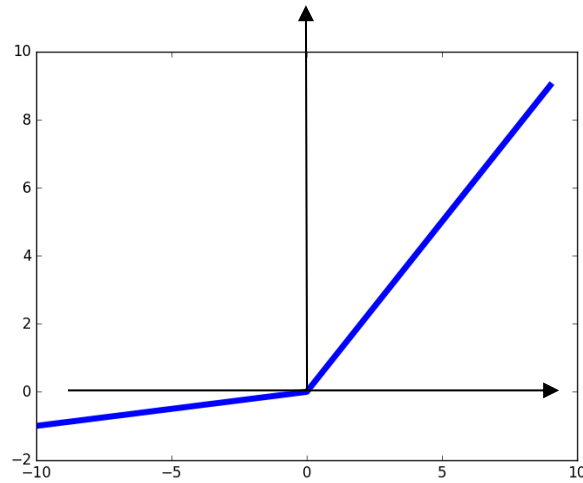
Leaky ReLU(x)

- no gradient saturation 😊
- Super fast 😊
- 0.01 is a hyperparameter 😊

[Mass et al., 2013]
[He et al., 2015]

Activation functions

$$\text{PReLU}(x) = \max(\alpha x, x)$$



Parametric ReLU(x)

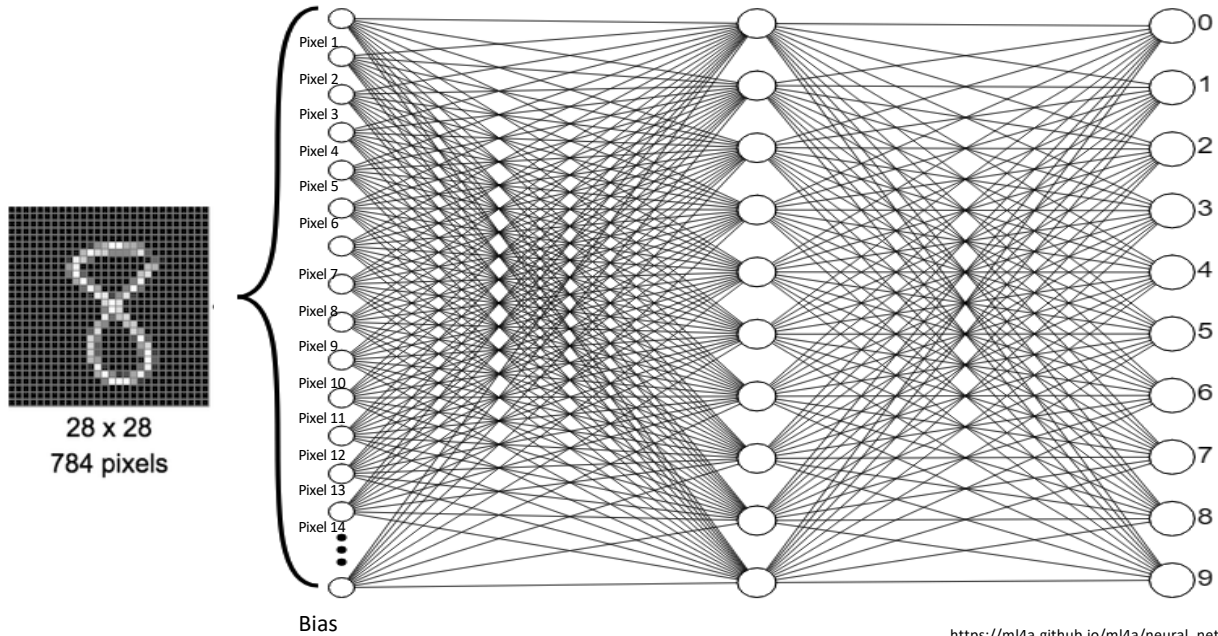
- no gradient saturation 😊
- Super fast 😊
- α learned with backprop 😊

[Mass et al., 2013]
[He et al., 2015]

In practice

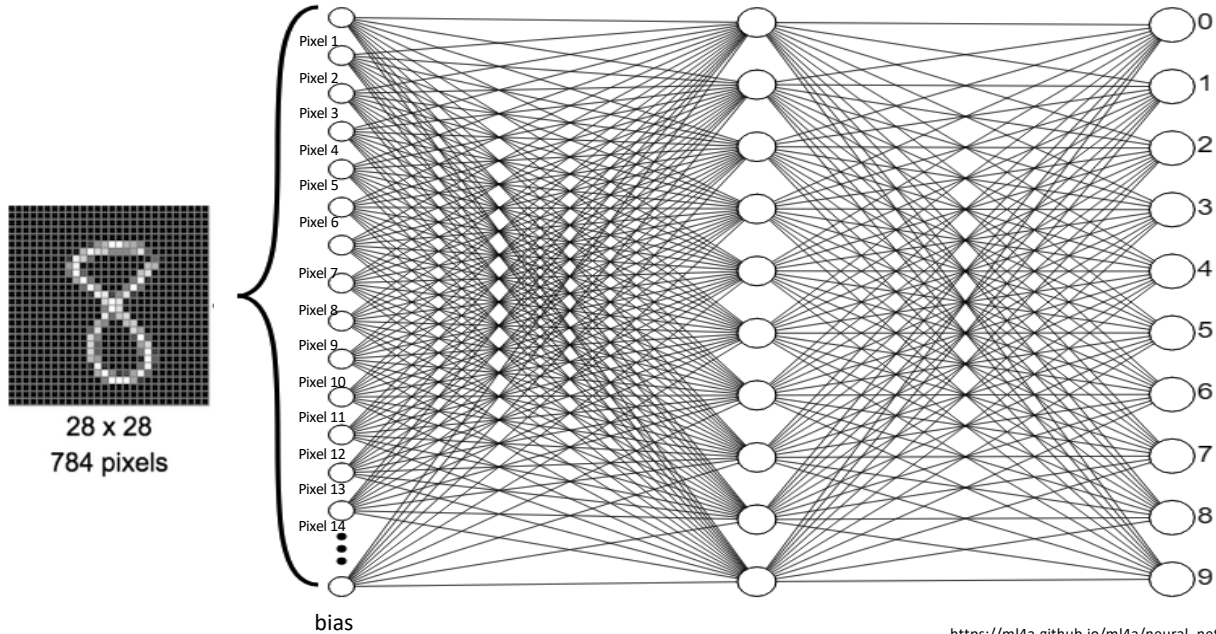
- By default, people use **ReLU**.
- Try **Leaky ReLU / PReLU**
- Try **tanh** but might be sub-optimal
- **Do not use sigmoid** except at the output of a 2 class net.

How to classify an image?



https://ml4a.github.io/ml4a/neural_networks/

Many parameters (7850 in Layer 1)

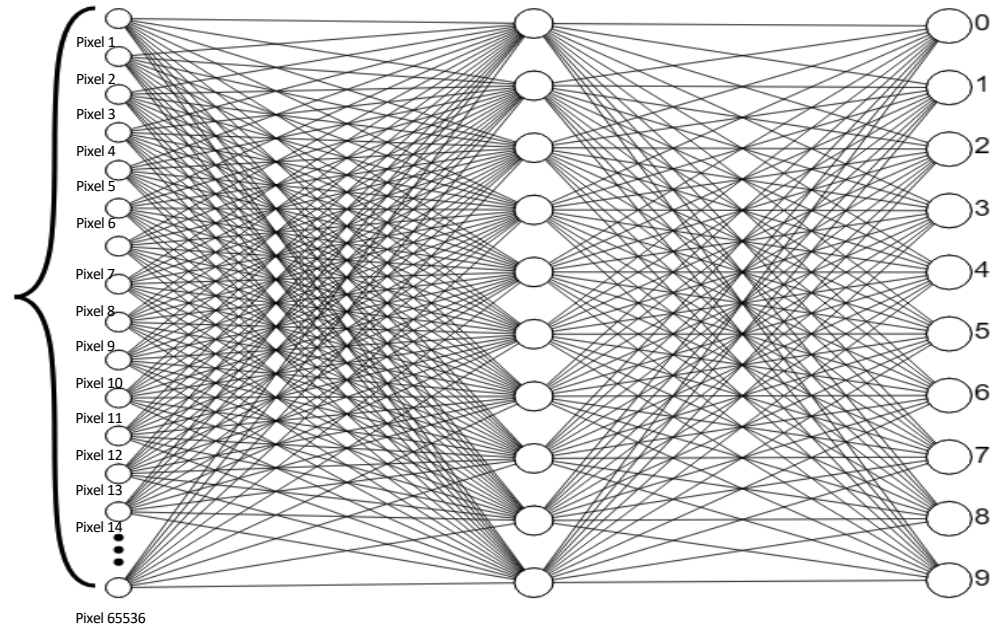


https://ml4a.github.io/ml4a/neural_networks/

Too many parameters (655,370 in Layer 1)

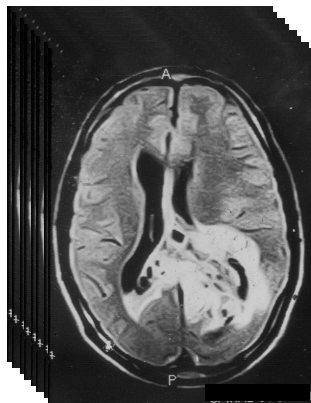


256x256

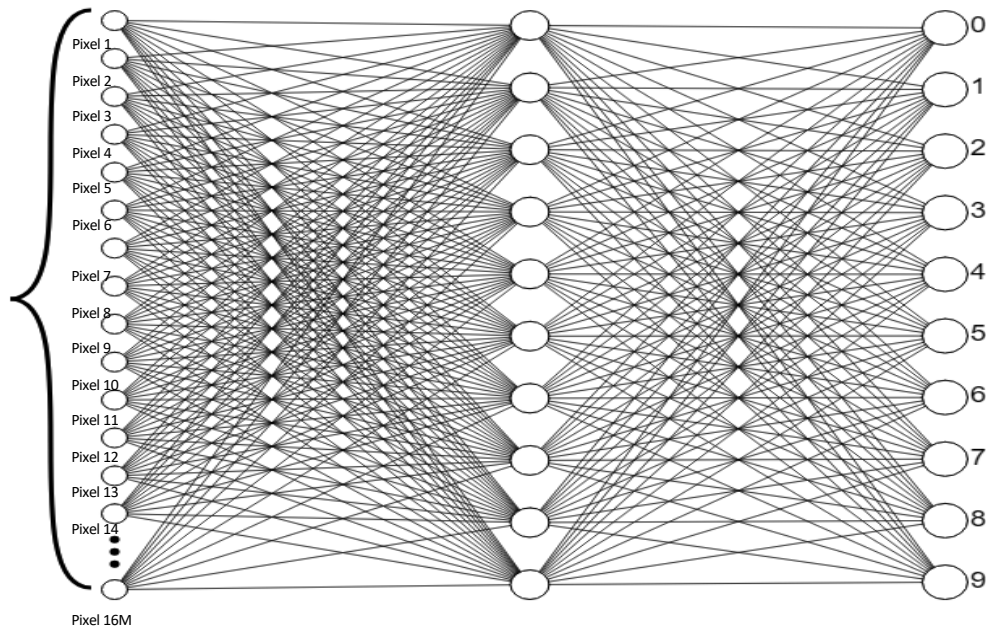


https://ml4a.github.io/ml4a/neural_networks/

Waaay too many parameters (160M in Layer 1)

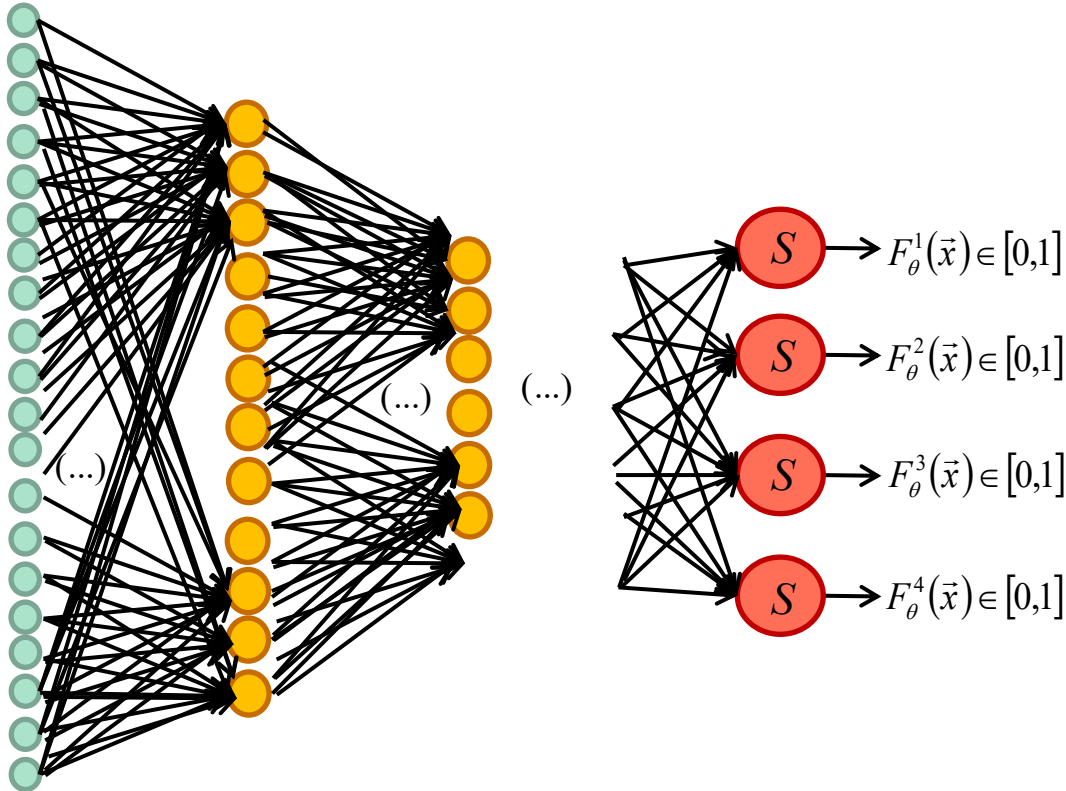


256x256x256



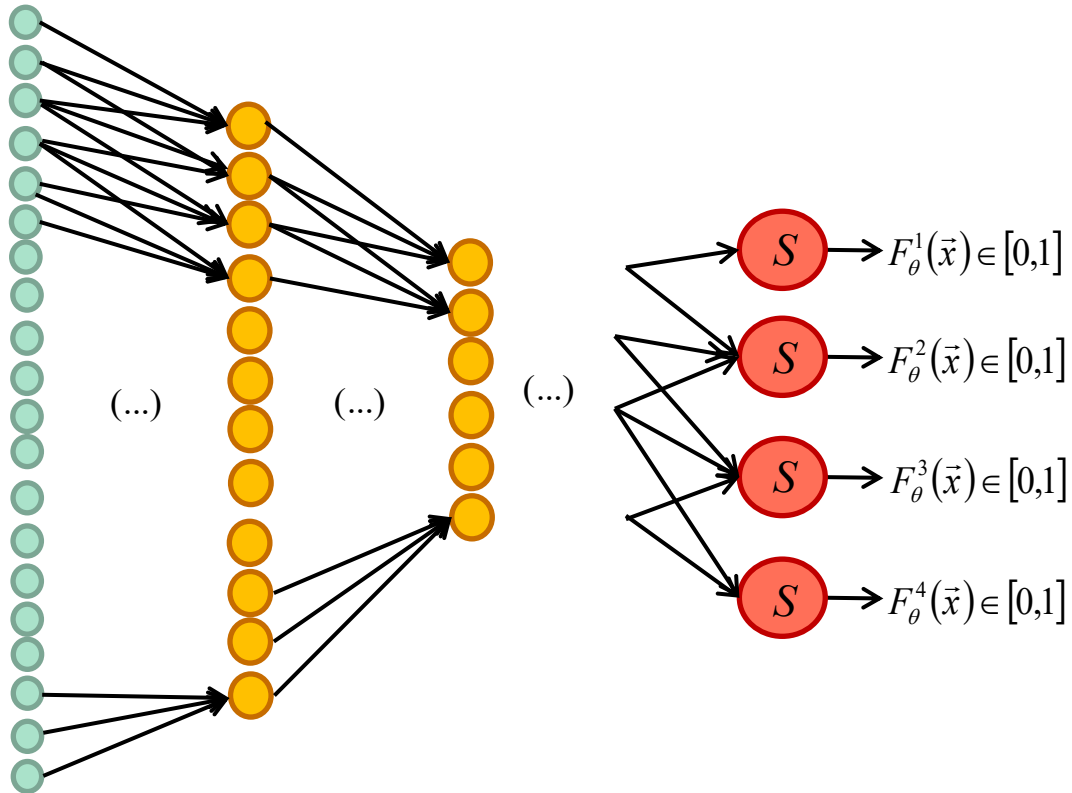
https://ml4a.github.io/ml4a/neural_networks/

Full connections are too many



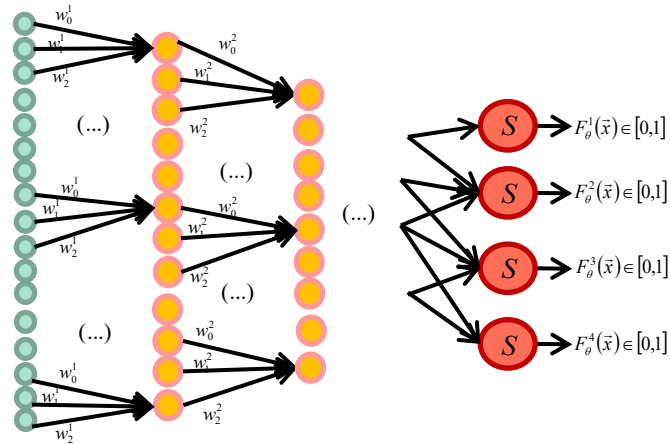
150-D input vector with 150 neurons in Layer 1 => 22,500 parameters !

No full connection



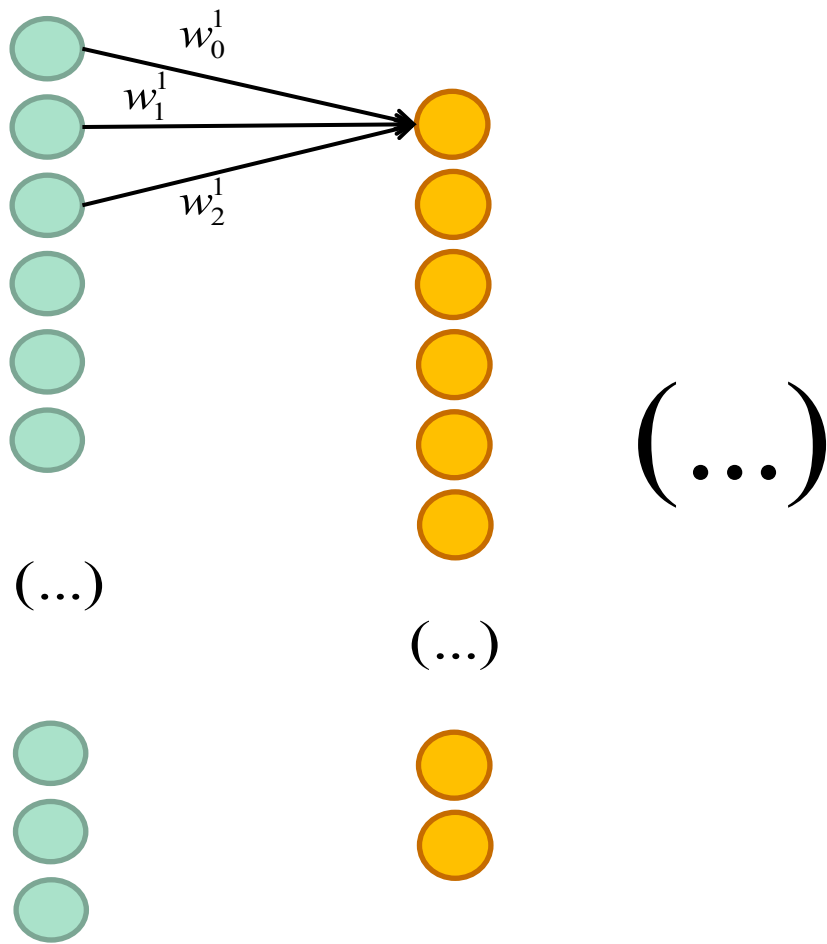
150-D input vector with 150 neurons in Layer 1 => 450 parameters !

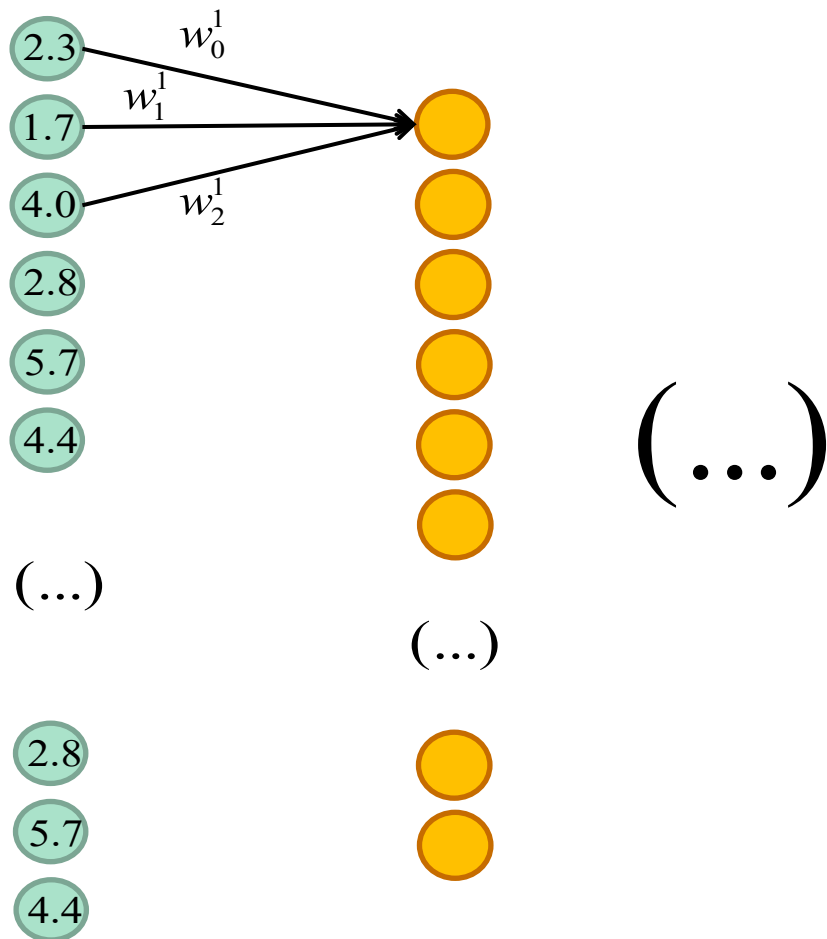
Share weights

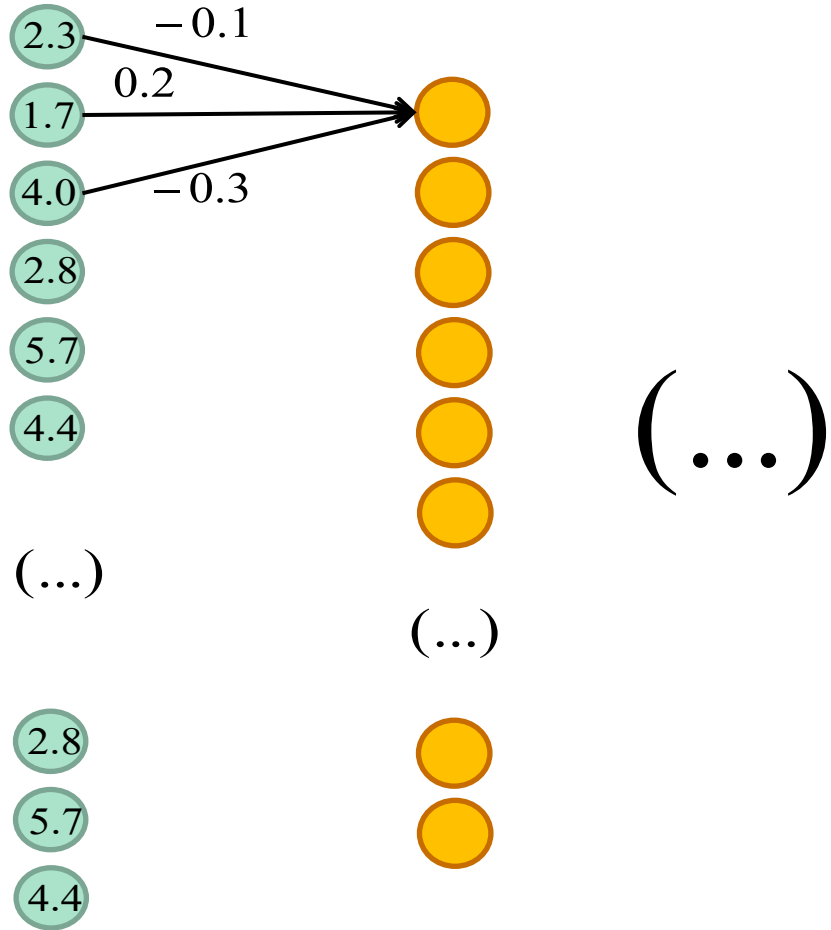


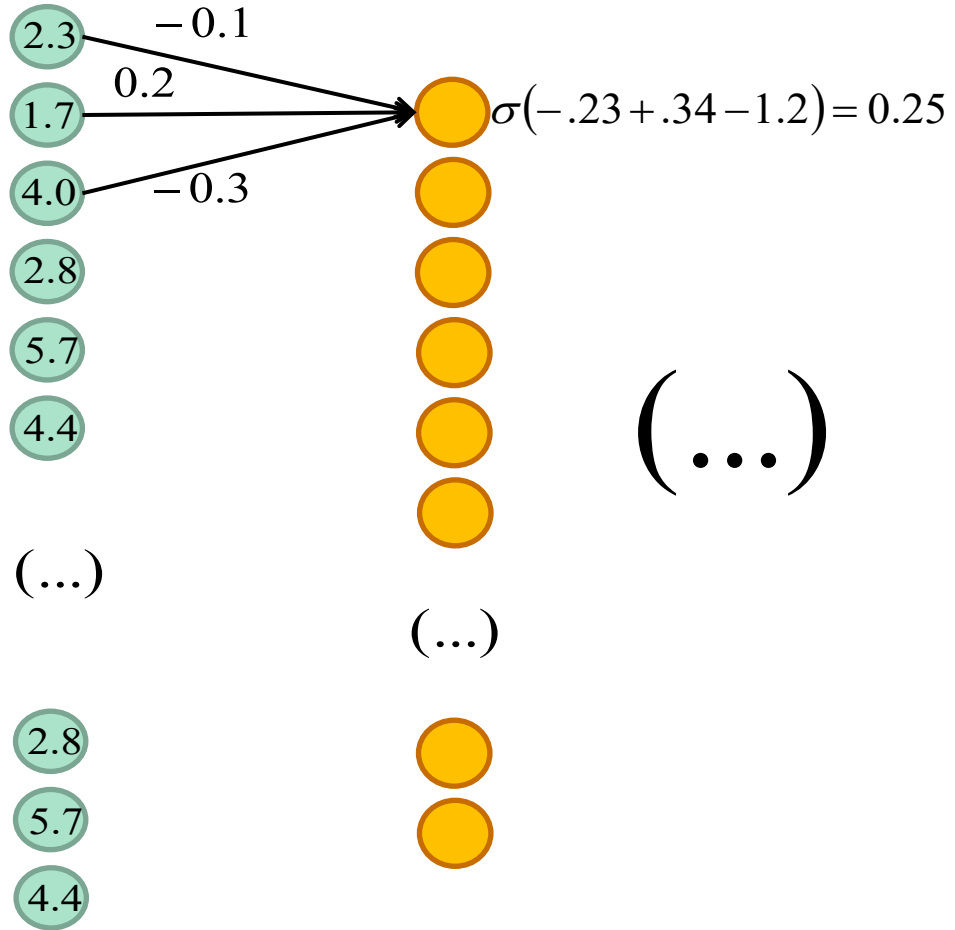
- 1- Learning convolution filters!
- 2- Small number of parameters = can make it deep!

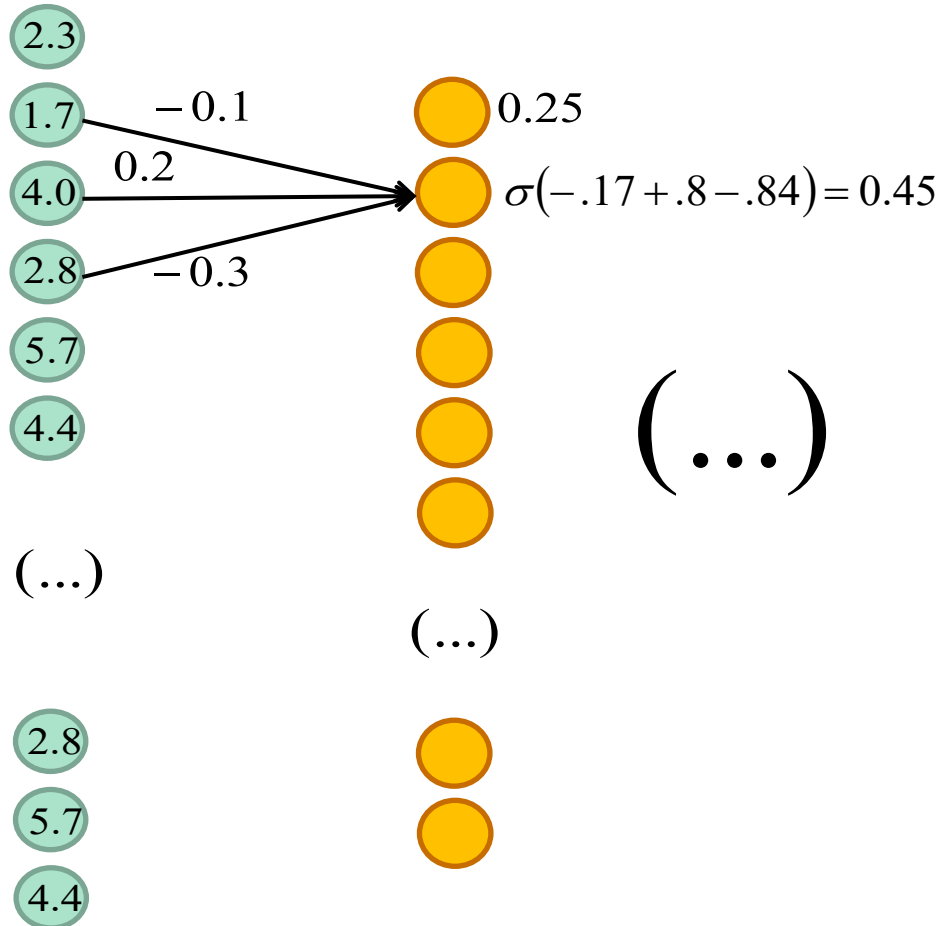
150-D input vector with 150 neurons in Layer 1 => 3 parameters !

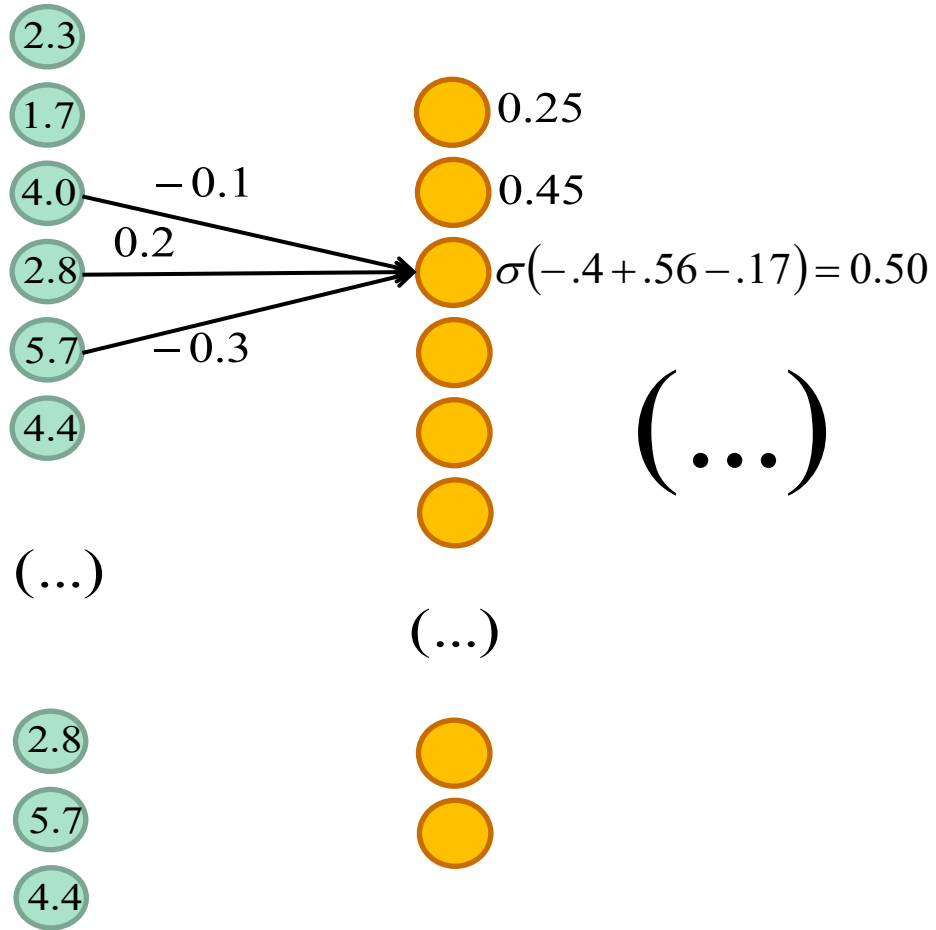












2.3

1.7

4.0

2.8

5.7

4.4

(...)

2.8

5.7

4.4

0.25

0.45

0.50

(...)

(...)

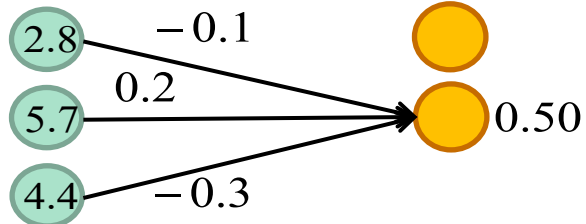
(...)

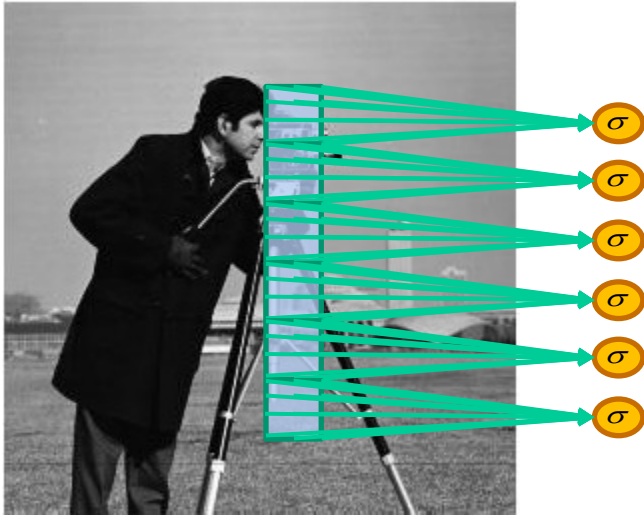
(...)

(...)

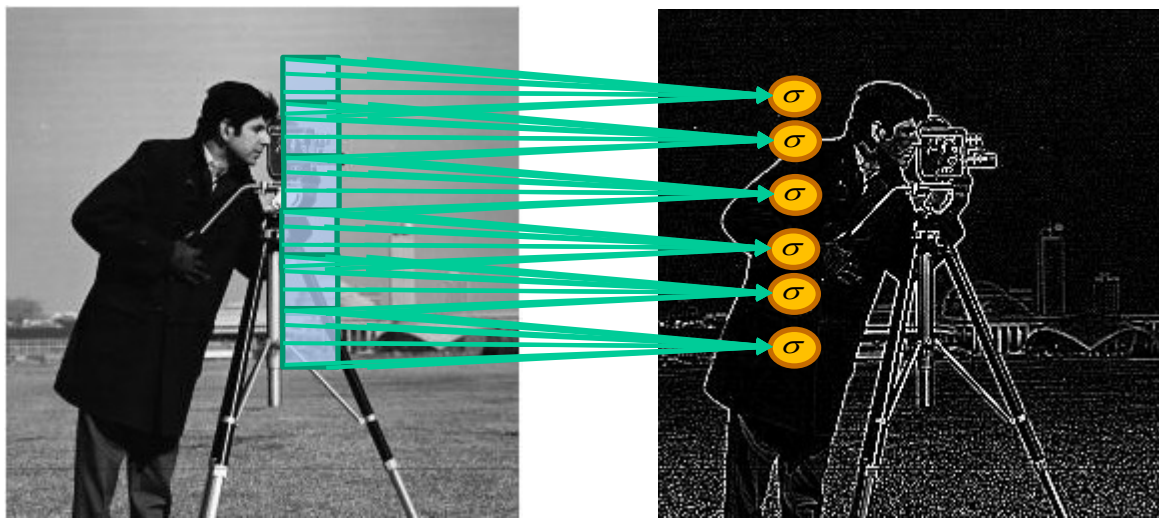
0.50

(...)





Each neuron of layer 1 is connected to 3 x 3 pixels
Layer 1 has only 9 parameters!!

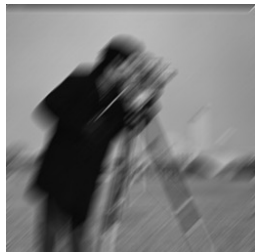


Convolution operation

Feature map



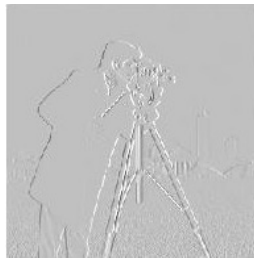
$$F = \sigma(x * W^{[0]})$$



$$\sigma(x * W_0^{[0]})$$



$$\sigma(x * W_4^{[0]})$$



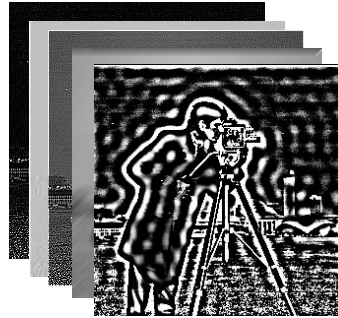
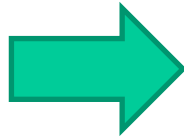
$$\sigma(x * W_1^{[0]})$$

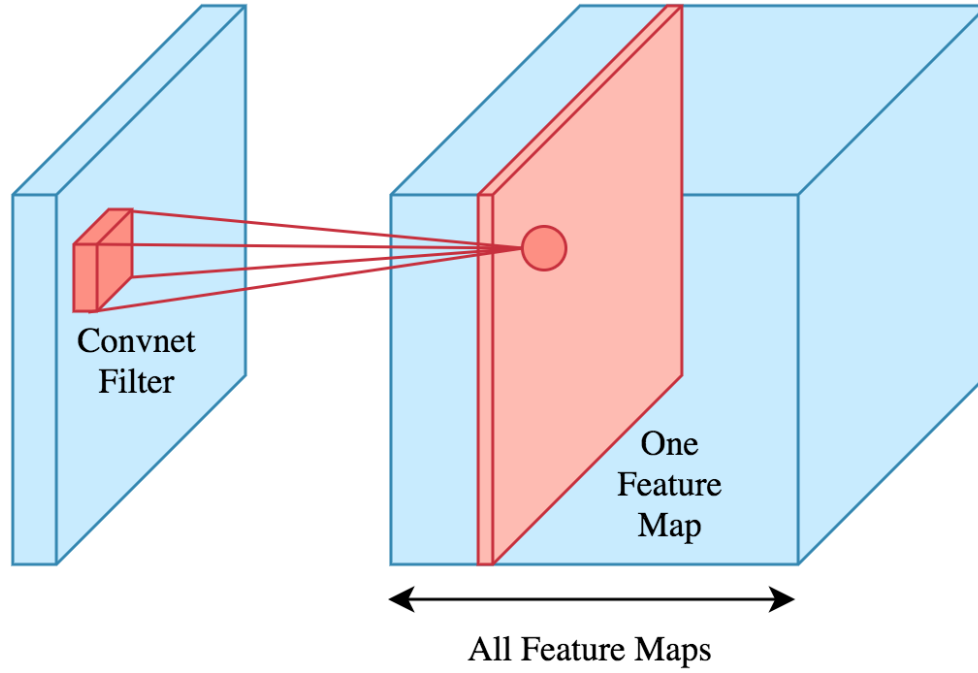


$$\sigma(x * W_3^{[0]})$$

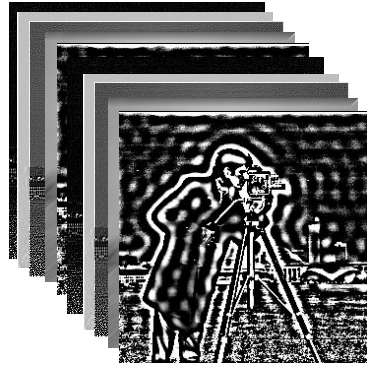
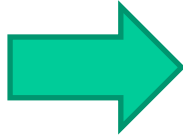


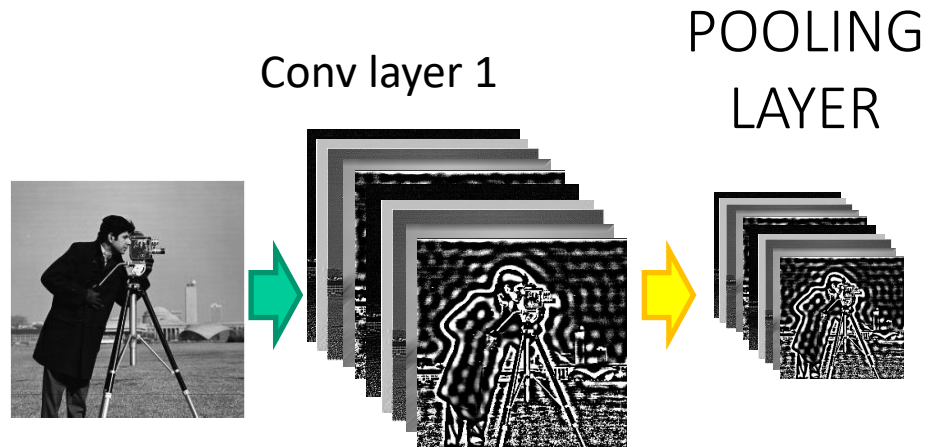
5-feature map convolution layer



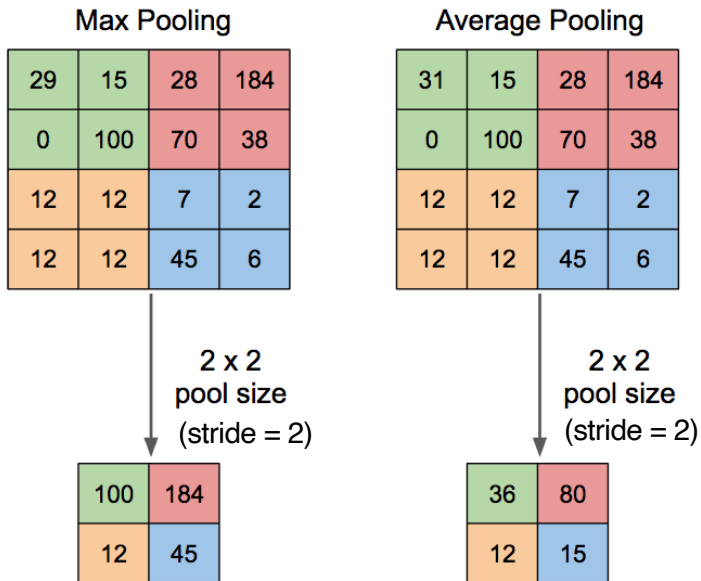


K-feature map convolution layer



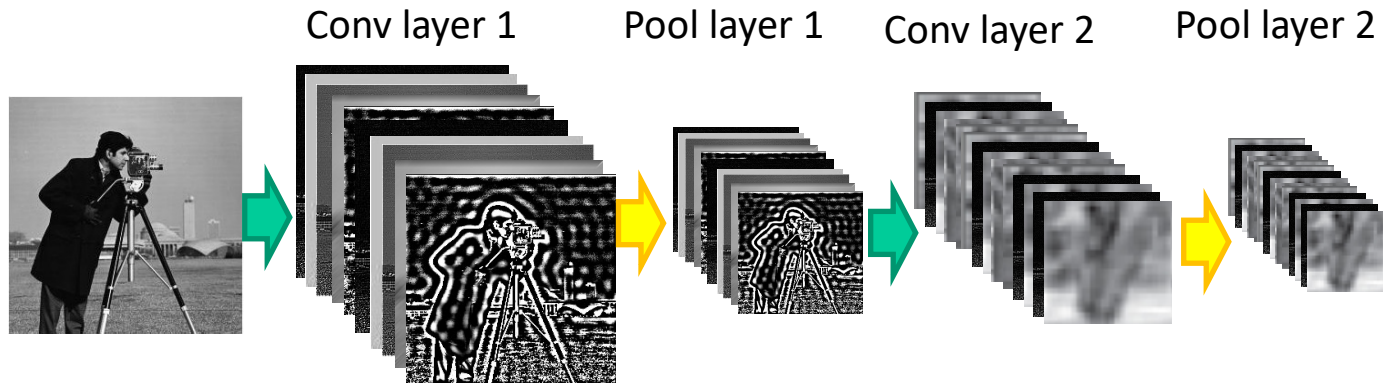


Pooling layer

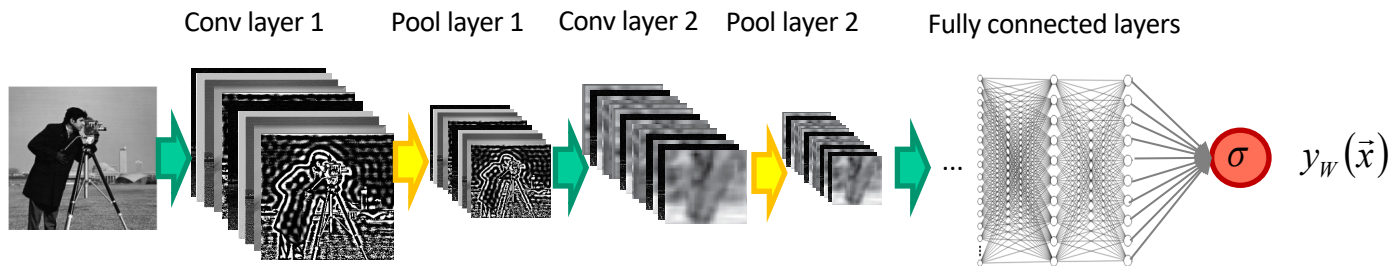


Goals

- Reduce the spatial resolution of feature maps
- Lower memory and computation requirements
- Provide partial invariance to position, scale and rotation

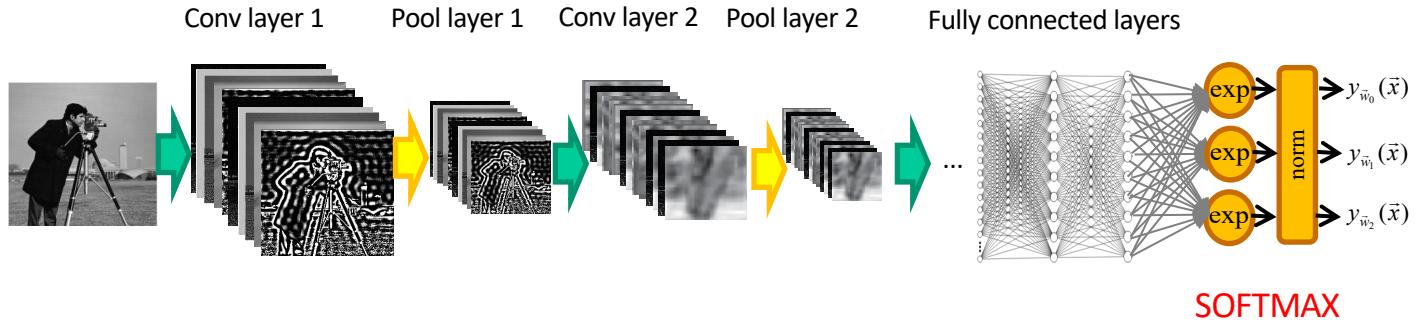


2 Class CNN



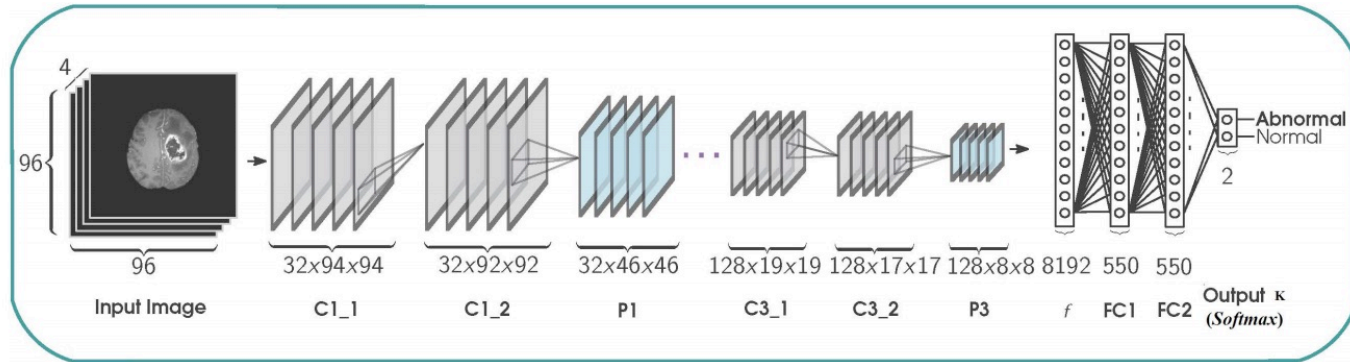
$$l(y_w(\vec{x}), t) = -t \ln(y_w(\vec{x})) - (1-t) \ln(1 - y_w(\vec{x}))$$

K Class CNN



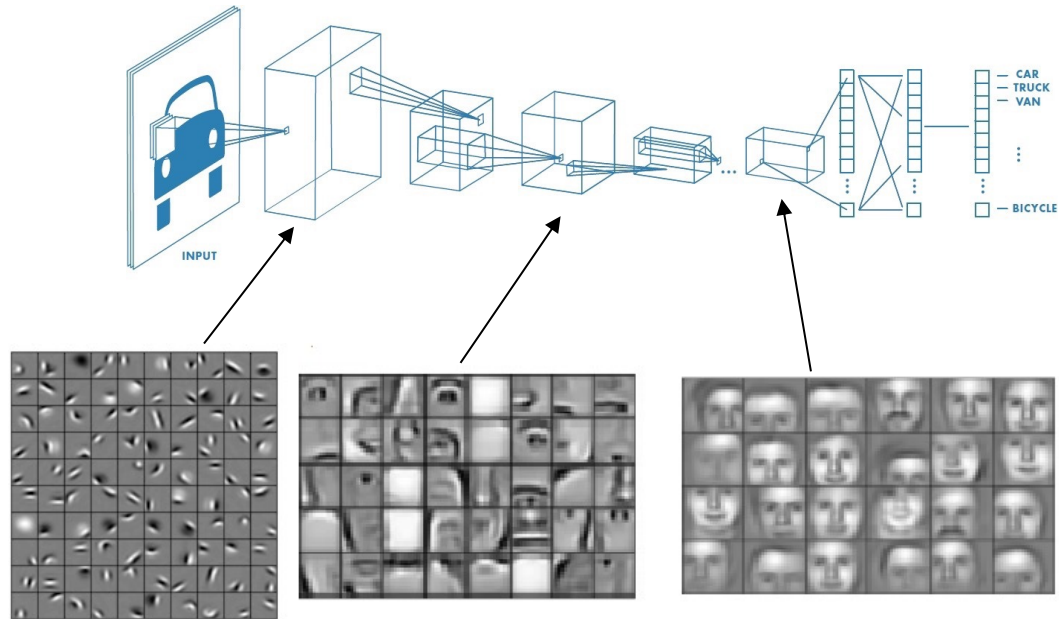
$$l(y_W(\vec{x}), t) = -t \ln(y_W(\vec{x})) - (1-t) \ln(1 - y_W(\vec{x}))$$

Nice example from the litterature



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

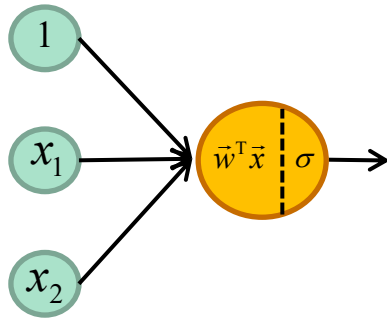
Learn image-based characteristics



<http://web.eecs.umich.edu/~honglak/icml09-ConvolutionalDeepBeliefNetworks.pdf>

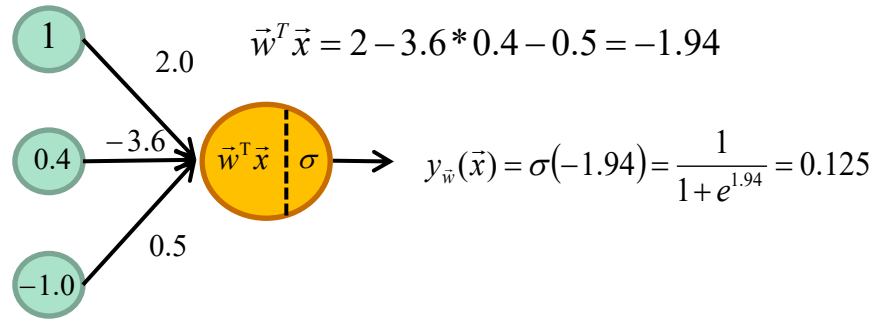
Batch processing

$$\vec{x} = (0.4, -1.0)$$
$$\vec{w} = [2.0, -3.6, 0.5]$$

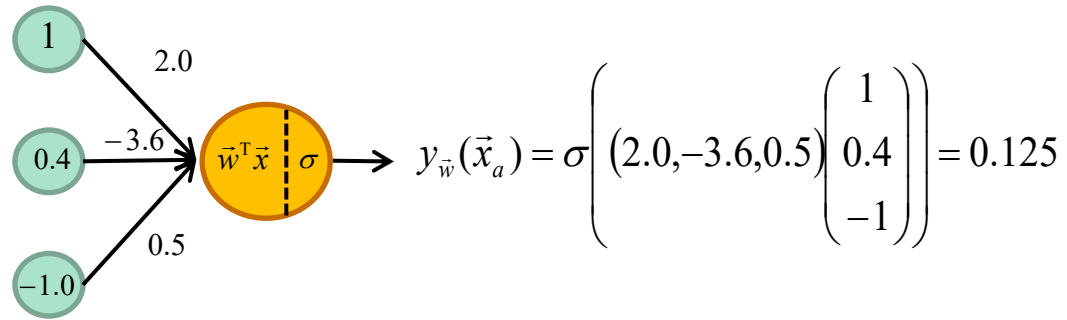


$$\vec{x} = (0.4, -1.0)$$

$$\vec{w} = [2.0, -3.6, 0.5]$$

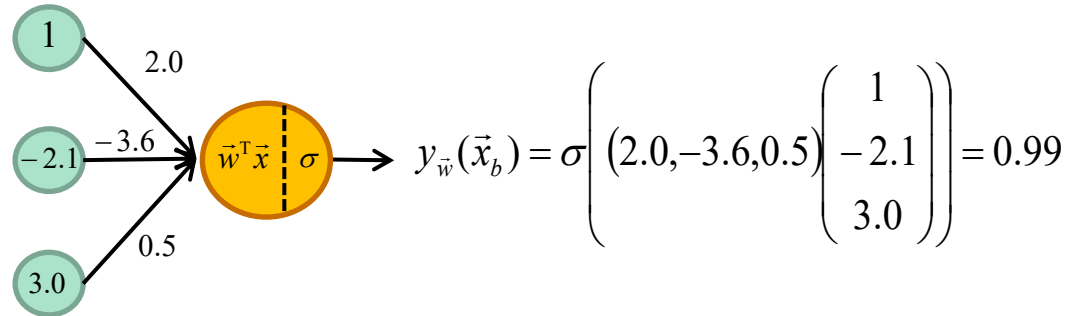
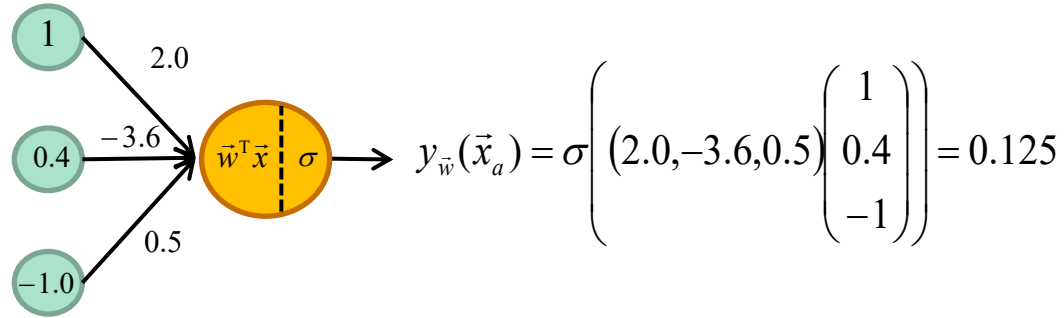


$$\vec{x} = (0.4, -1.0)$$
$$\vec{w} = [2.0, -3.6, 0.5]$$

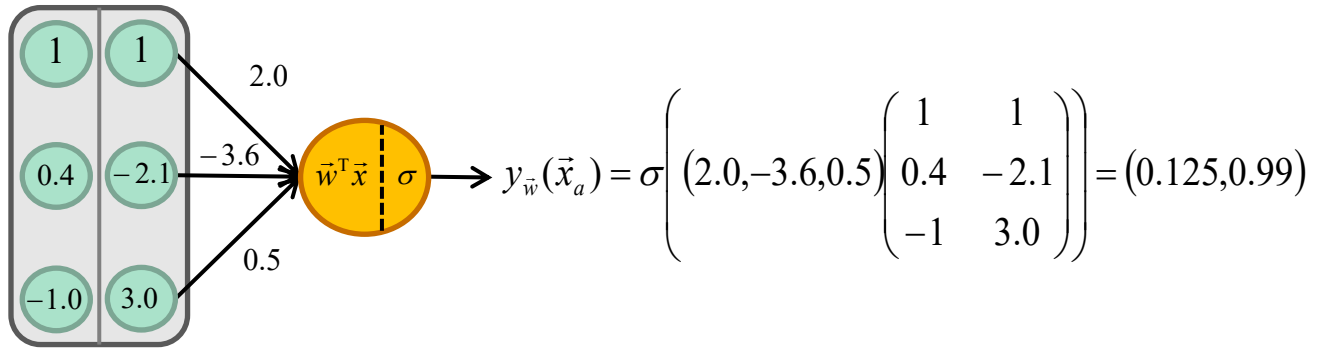


$$\vec{x}_a = (0.4, -1.0), \vec{x}_b = (-2.1, 3.0)$$

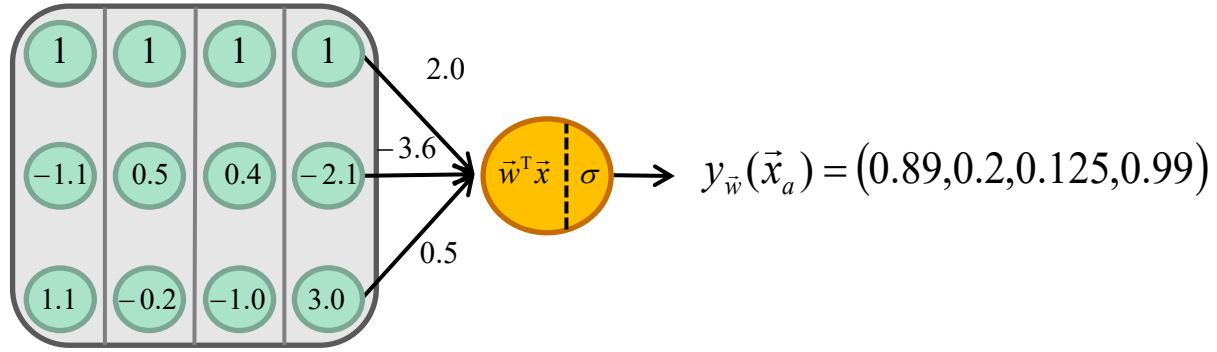
$$\vec{w} = [2.0, -3.6, 0.5]$$



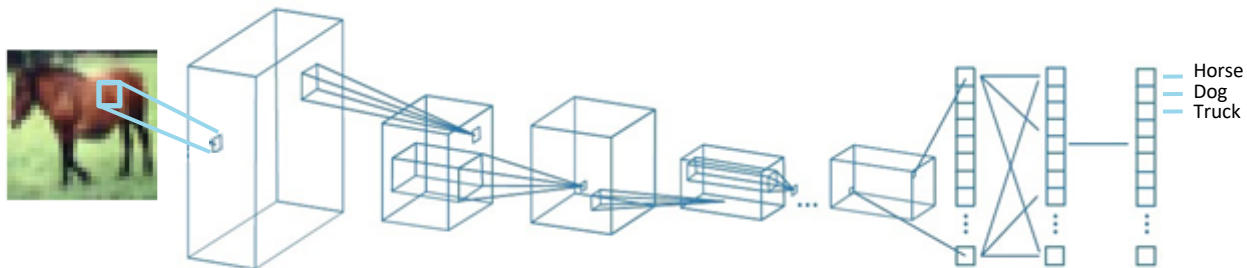
Mini-batch processing



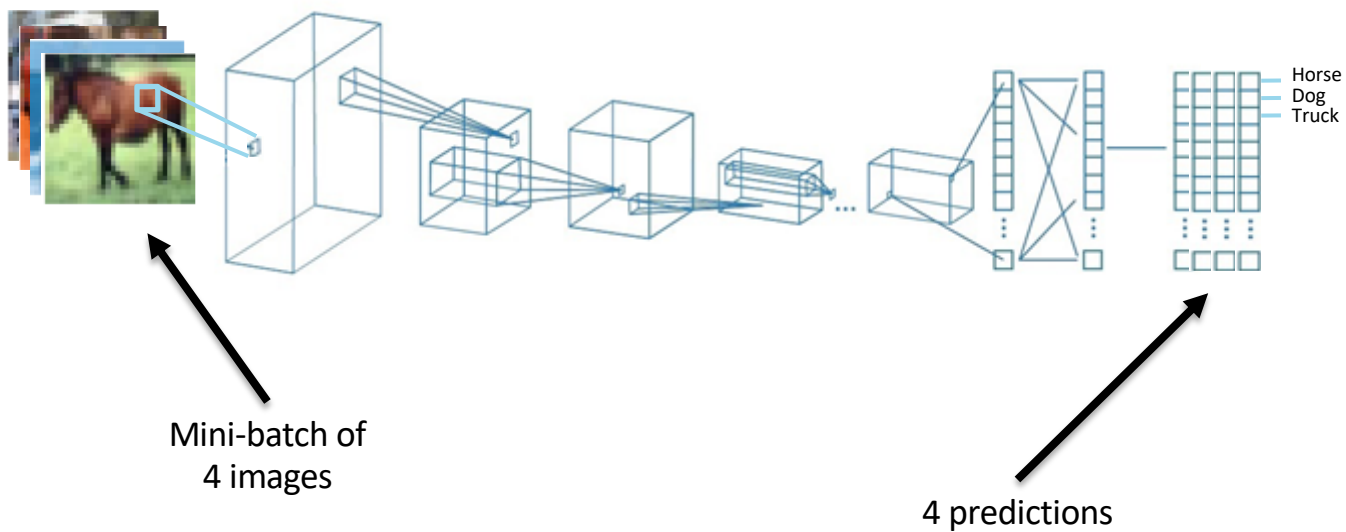
Mini-batch processing



Mini-batch processing

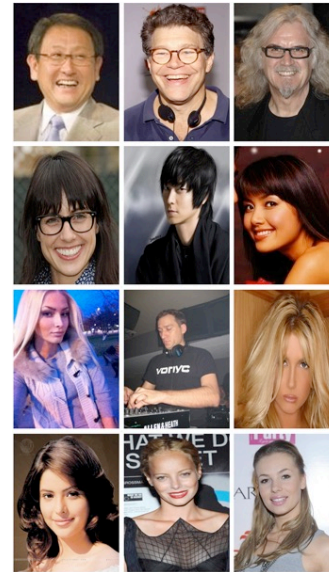


Mini-batch processing



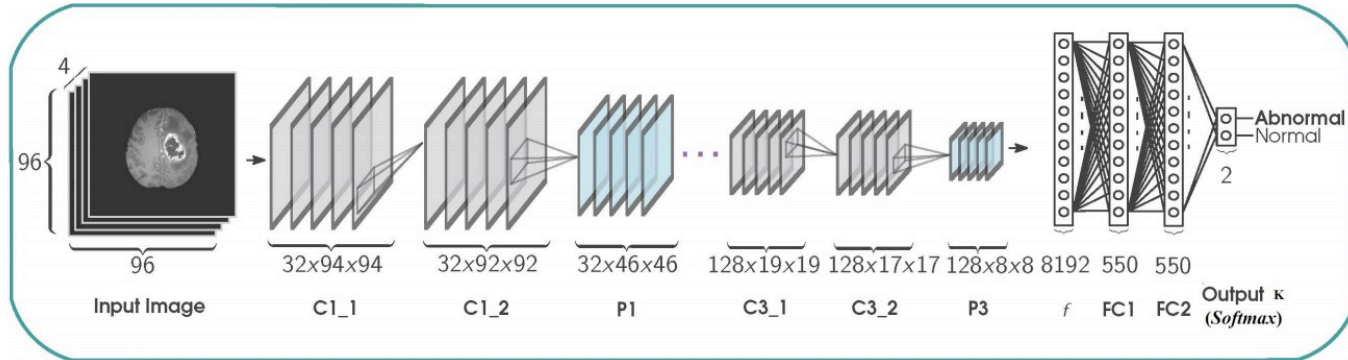
Classical applications of ConvNets

Classification



Classical applications of ConvNets

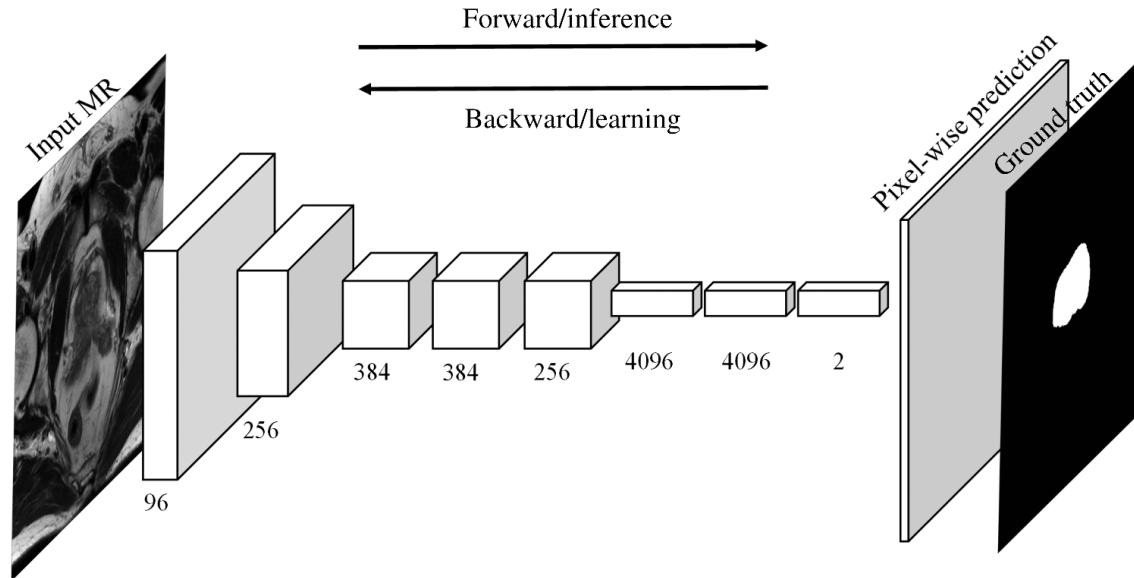
Classification



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

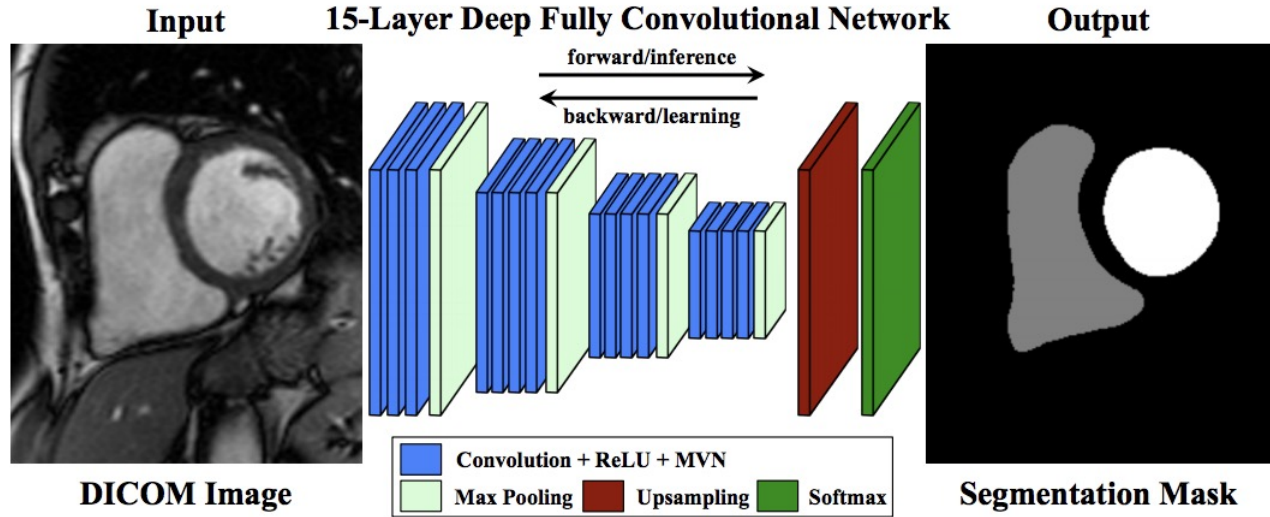
Classical applications of ConvNets

Image segmentation



Classical applications of ConvNets

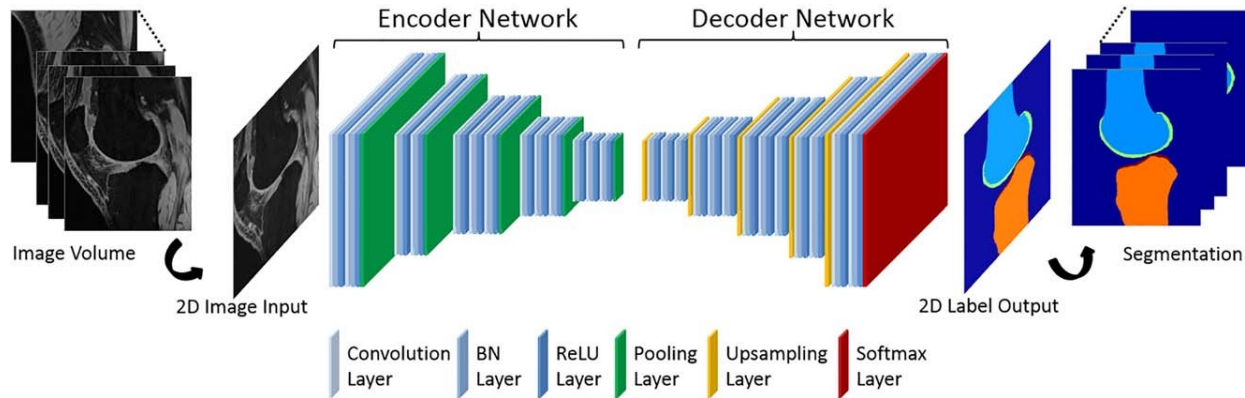
Image segmentation



Tran, P. V., 2016. A fully convolutional neural network for cardiac segmentation in short-axis MRI. arXiv:1604.00494.

Classical applications of ConvNets

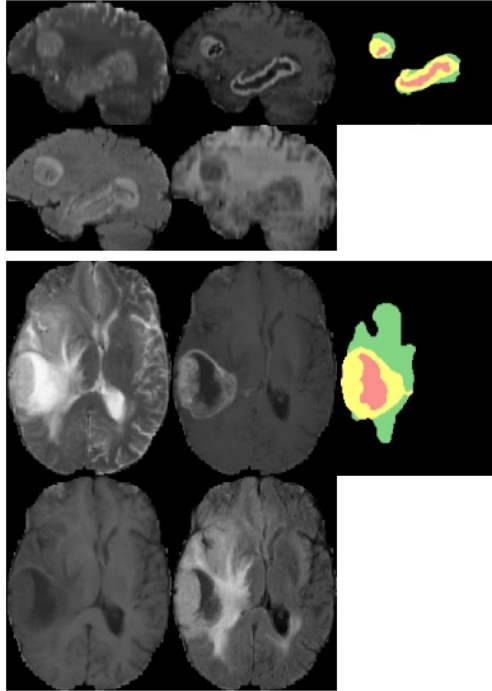
Image segmentation



Fang Liu, Zhaoye Zhou, +3 authors, Deep convolutional neural network and 3D deformable approach for tissue segmentation in musculoskeletal magnetic resonance imaging. in Magnetic resonance in medicine 2018 DOI:10.1002/mrm.26841

Classical applications of ConvNets

Image segmentation



Havaei M., Davy A., Warde-Farley D., Biard A., Courville A., Bengio Y., Pal C., Jodoin P-M, Larochelle H. (2017)
Brain Tumor Segmentation with Deep Neural Networks, Medical Image Analysis, Vol 35, 18-31

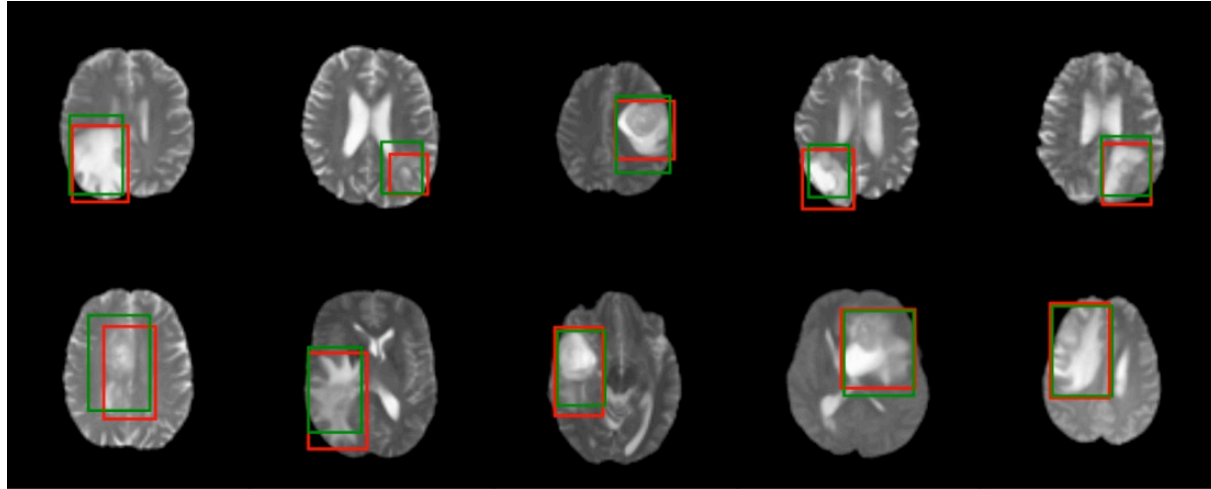
Classical applications of ConvNets

Localization



Classical applications of ConvNets

Localization



S. Banerjee, S. Mitra, A. Sharma, and B. U Shankar, A CADe System for Gliomas in Brain MRI using Convolutional Neural Networks, arXiv:1806.07589v, 2018

Merçi

Big thanks to



Pierre-Marc Jodoin



Université de
Sherbrooke