Deep Learning

Instructor - Simon Lucey

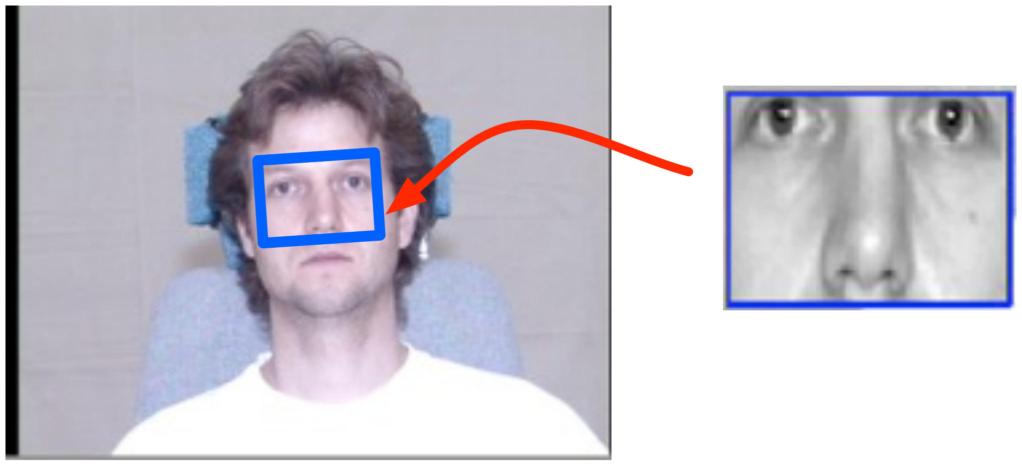
16-423 - Designing Computer Vision Apps

Carnegie Mellon
THE ROBOTICS INSTITUTE

Today

- Single-Layer Perceptron
- Multi-Layer Perceptron
- Convolutional Neural Network

Linear Binary Classification



$$[65,09,67,\ldots\ldots,78,66,76,215]^T$$

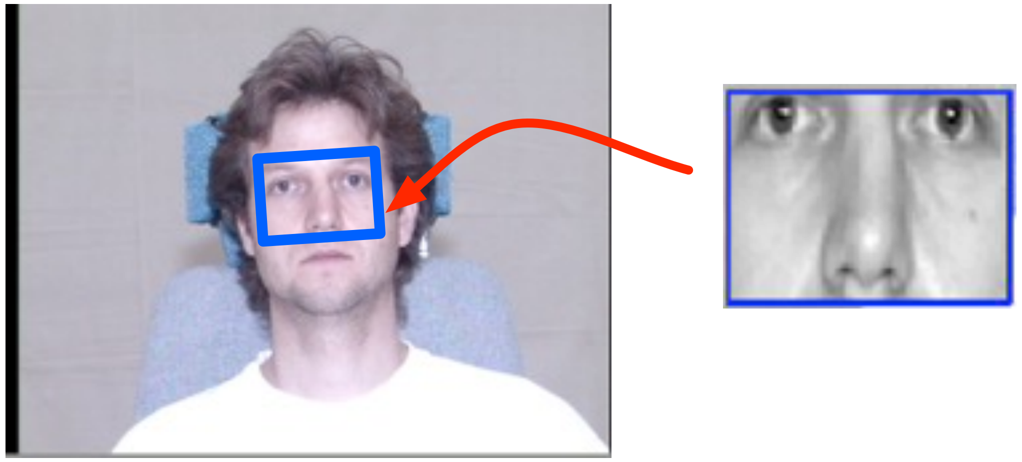
$$\mathbf{x} \in \mathbb{R}^D$$

$$\mathbf{x} \in \mathbb{C}_1$$

$$\mathbf{w}^T\mathbf{x} + w_0 \underset{<}{\overset{>}{-}} 0$$

$$\mathbf{x} \in \mathbb{C}_2$$

Linear Binary Classification



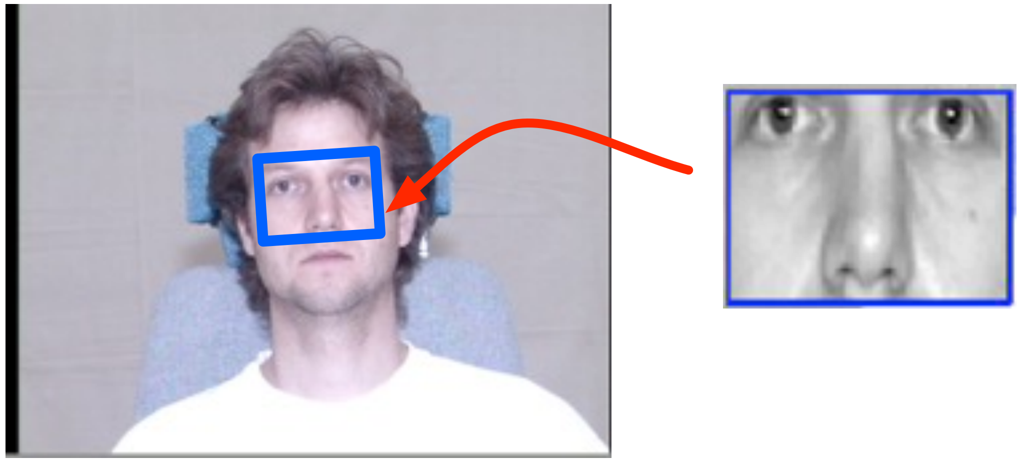
$$[65,09,67,\ldots\ldots..,78,66,76,215]^T$$

$$\mathbf{x} \in \mathbb{R}^D$$

“Perceptron”

$$\mathbf{w}^T\mathbf{x} + w_0 \underset{\mathbf{x} \in \mathbb{C}_2}{\overset{\mathbf{x} \in \mathbb{C}_1}{\gtrless}} 0$$

Linear Binary Classification



$$[65,09,67,\ldots\ldots,78,66,76,215]^T$$

$$\mathbf{x} \in \mathbb{R}^D$$

"Linear Discriminant"

$$\mathbf{x} \in \mathbb{C}_1$$

$$\boxed{\mathbf{w}^T\mathbf{x} + w_0} \gtrless 0$$

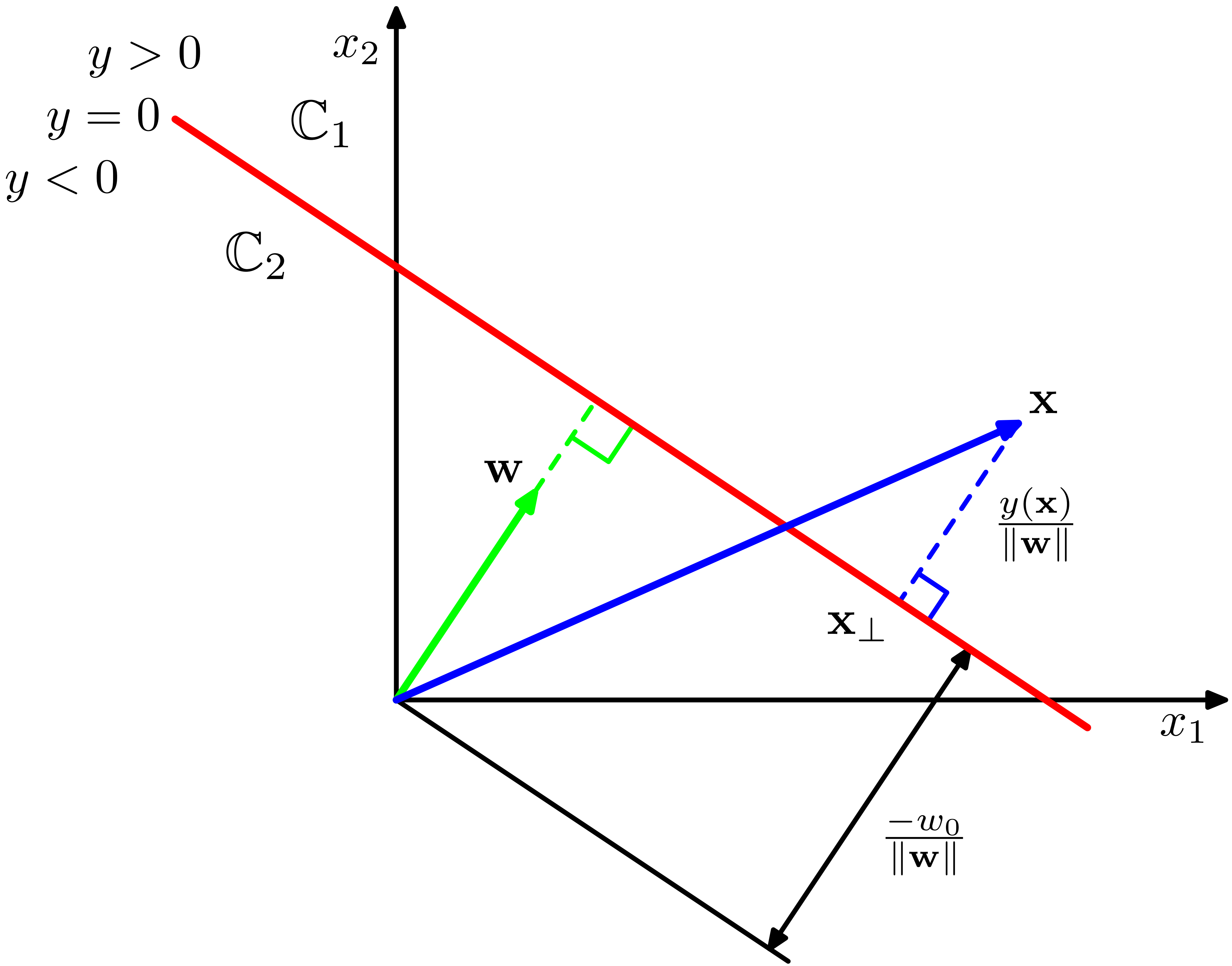
$$\mathbf{x} \in \mathbb{C}_2$$

Reminder: Perceptron

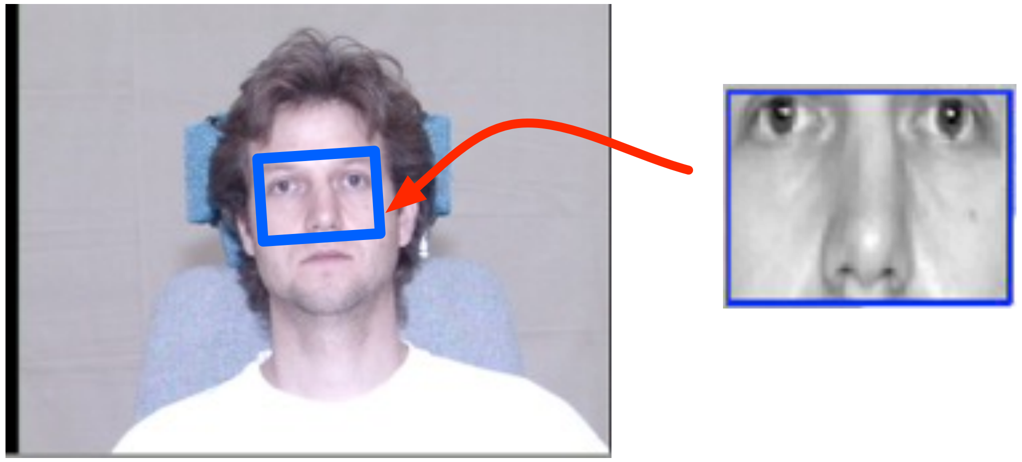
- Rosenblatt simulated the perceptron on a IBM 704 computer at Cornell in 1957.
- Input scene (i.e. printed character) was illuminated by powerful lights and captured on a 20x20 cadmium sulphide photo cells.
- Weights of perceptron were applied using variable rotary resistors.
- Often times referred to as the very first neural network.

"Frank Rosenblatt"

Linear Discriminant Functions



Linear Binary Classification

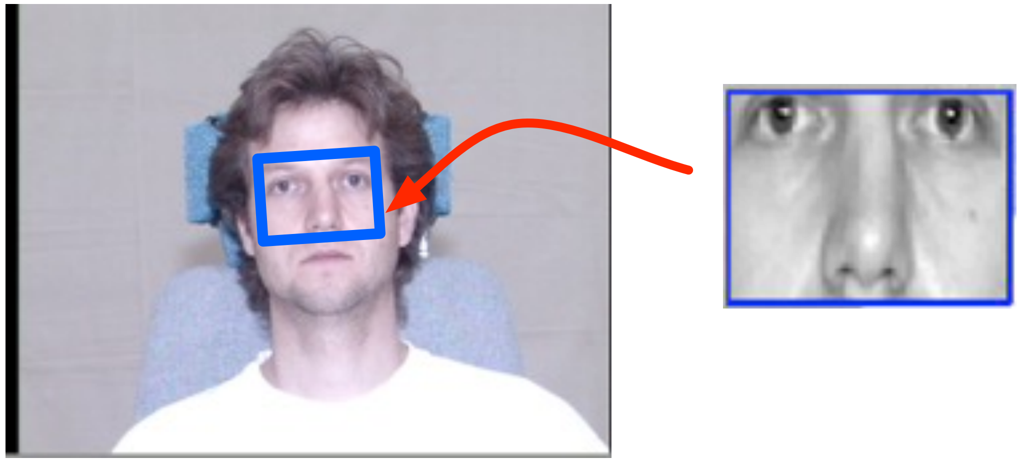


$$[65,09,67,\ldots\ldots,78,66,76,215]^T$$

$$\mathbf{x} \in \mathbb{R}^D$$

$$\begin{bmatrix}\mathbf{w}\\ w_0\end{bmatrix}^T \begin{bmatrix}\mathbf{x}\\ 1\end{bmatrix} \underset{\mathbf{x}\in\mathbb{C}_2}{\overset{\mathbf{x}\in\mathbb{C}_1}{\substack{\geq\\ <}}} 0$$

Linear Binary Classification

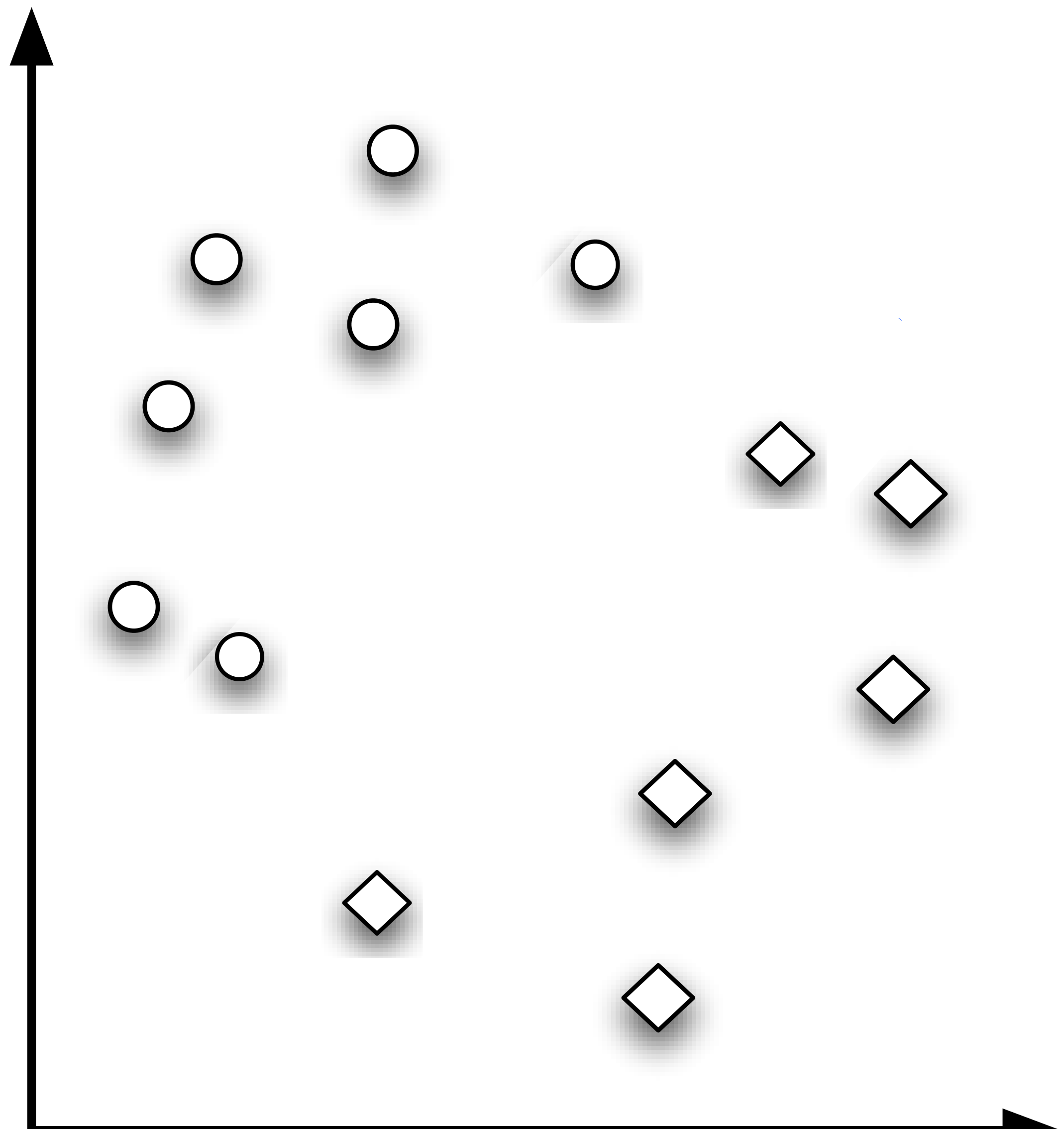


$$[65,09,67,........,78,66,76,215]^T$$

$$\mathbf{x} \in \mathbb{R}^D$$

$$\mathbf{w}^T\mathbf{x} \quad \begin{array}{c} \mathbf{x} \in \mathbb{C}_1 \\ \geq \\ < \\ \mathbf{x} \in \mathbb{C}_2 \end{array} \quad 0$$

Perceptron Linear Discriminant



○ $t_i = +1$

◇ $t_i = -1$

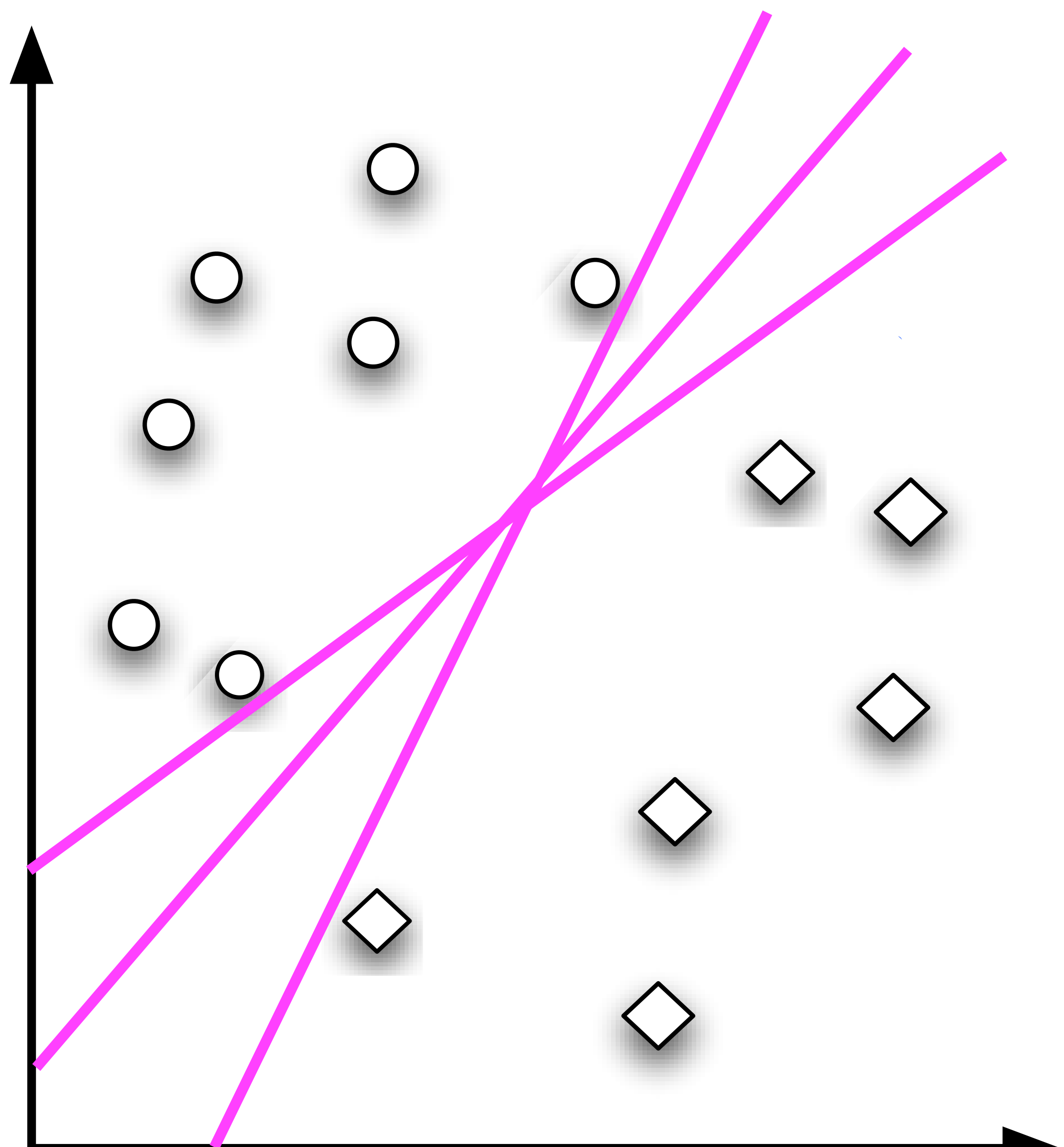
binary labels

$\mathbf{x}_i$ = i-th training example

$\mathbf{w}$ = weight vector

$$\arg\min_{\mathbf{w}} \sum_{n=1}^{N} \max(0, t_n \cdot \mathbf{x}_n^T \mathbf{w})$$

Perceptron Linear Discriminant



○ $t_i = +1$

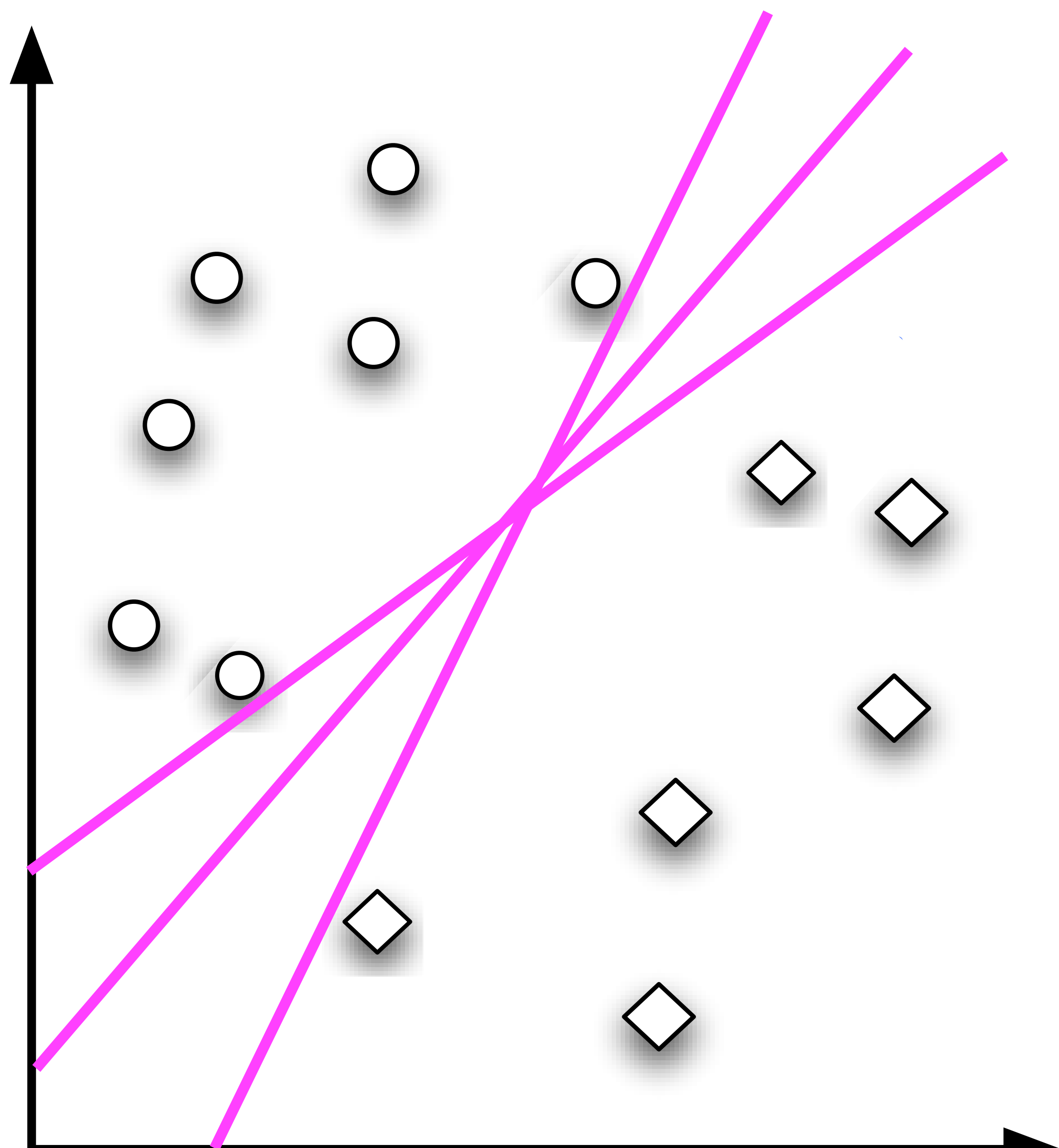
◇ $t_i = -1$

binary labels

$\mathbf{x}_i = i\text{-th training example}$
$\mathbf{w} = \text{weight vector}$

$$\arg\min_{\mathbf{w}} \sum_{n=1}^{N} \max(0, t_n \cdot \mathbf{x}_n^T \mathbf{w})$$

Perceptron Linear Discriminant



$$\arg\min_{\mathbf{w}} \sum_{n=1}^{N} E(t_n \cdot \mathbf{x}_n^T \mathbf{w})$$

○ $t_i = +1$

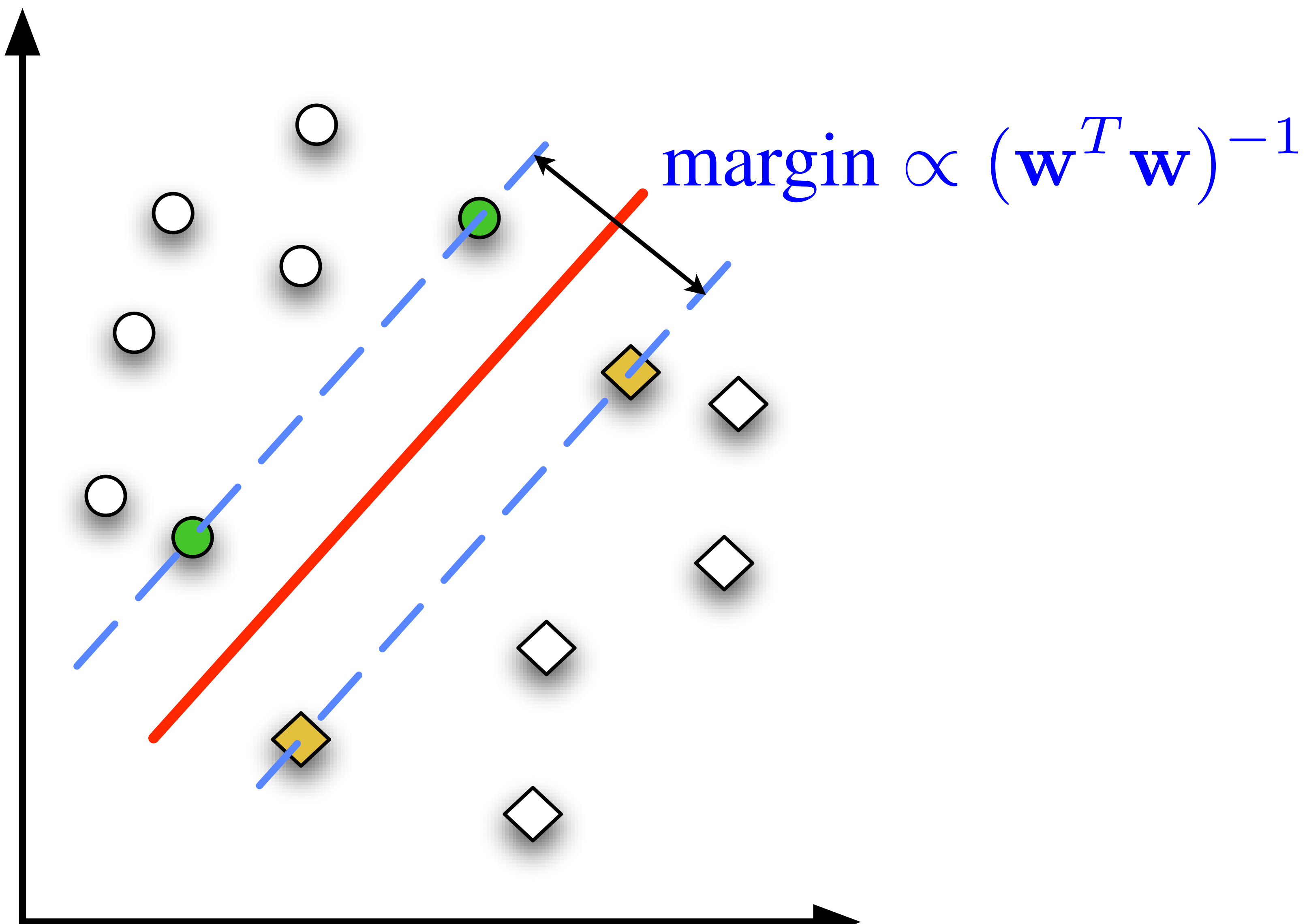
◇ $t_i = -1$

binary labels

$\mathbf{x}_i$ = i-th training example

$\mathbf{w}$ = weight vector

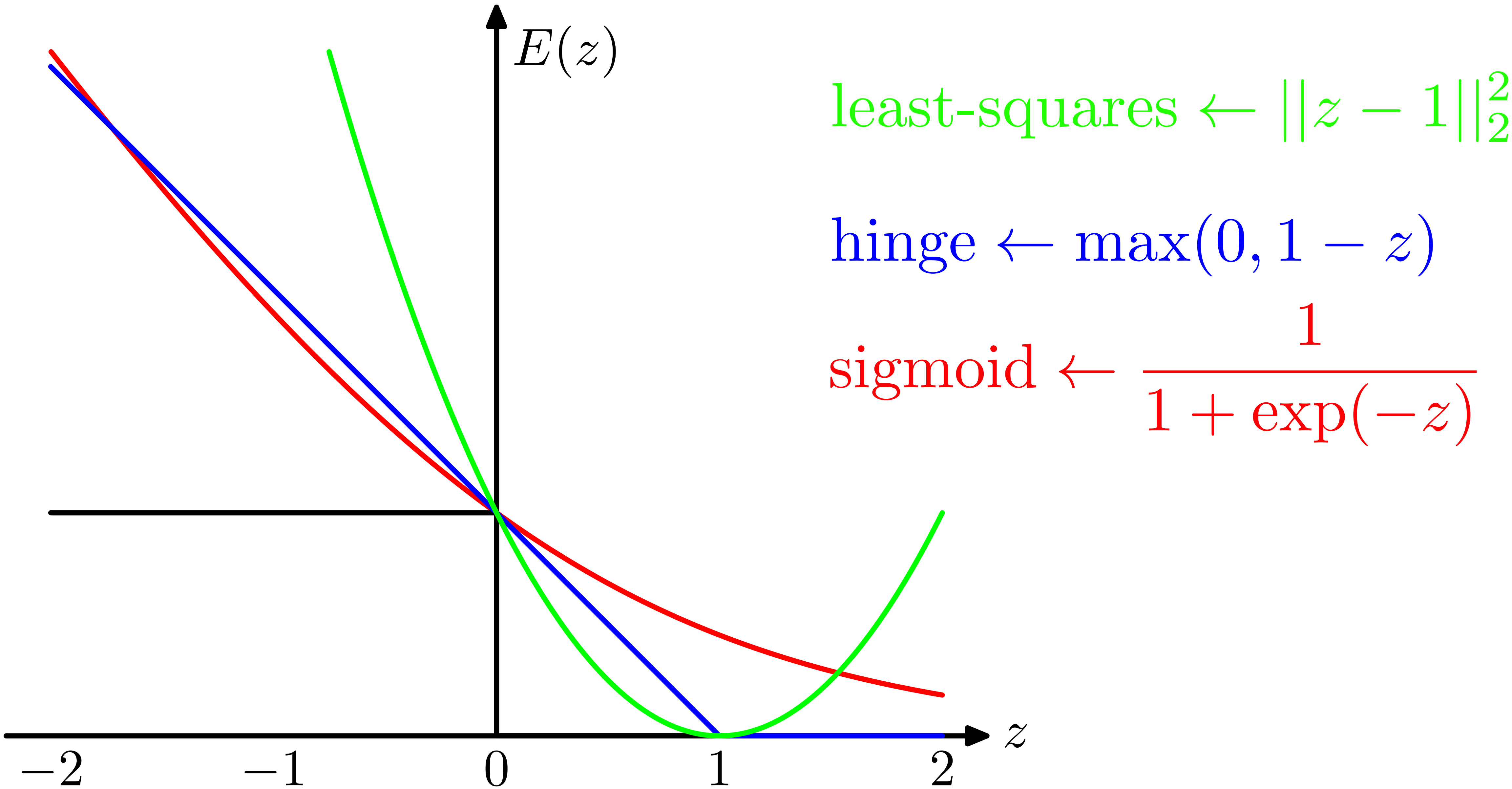
Perceptron Linear Discriminant



$$\arg\min_{\mathbf{w}} \sum_{n=1}^{N} E(t_n \cdot \mathbf{x}_n^T \mathbf{w}) + \frac{\lambda}{2}||\mathbf{w}||_2^2$$

Other Objectives

- Other objectives are possible,



Optimizing Weights

- Expressing the final objective as,

$$f(\mathbf{w}) = \sum_{n=1}^{N} E(t_n \cdot \mathbf{x}_n^T \mathbf{w})$$

- Simplest strategy is to employ gradient-descent optimization,

$$\mathbf{w} \rightarrow \mathbf{w} - \eta \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}$$

Optimizing Weights

- Expressing the final objective as,

$$f(\mathbf{w}) = \sum_{n=1}^{N} E(t_n \cdot \mathbf{x}_n^T \mathbf{w})$$

- Simplest strategy is to employ gradient-descent optimization,

$$\mathbf{w} \rightarrow \mathbf{w} - \eta \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}}$$

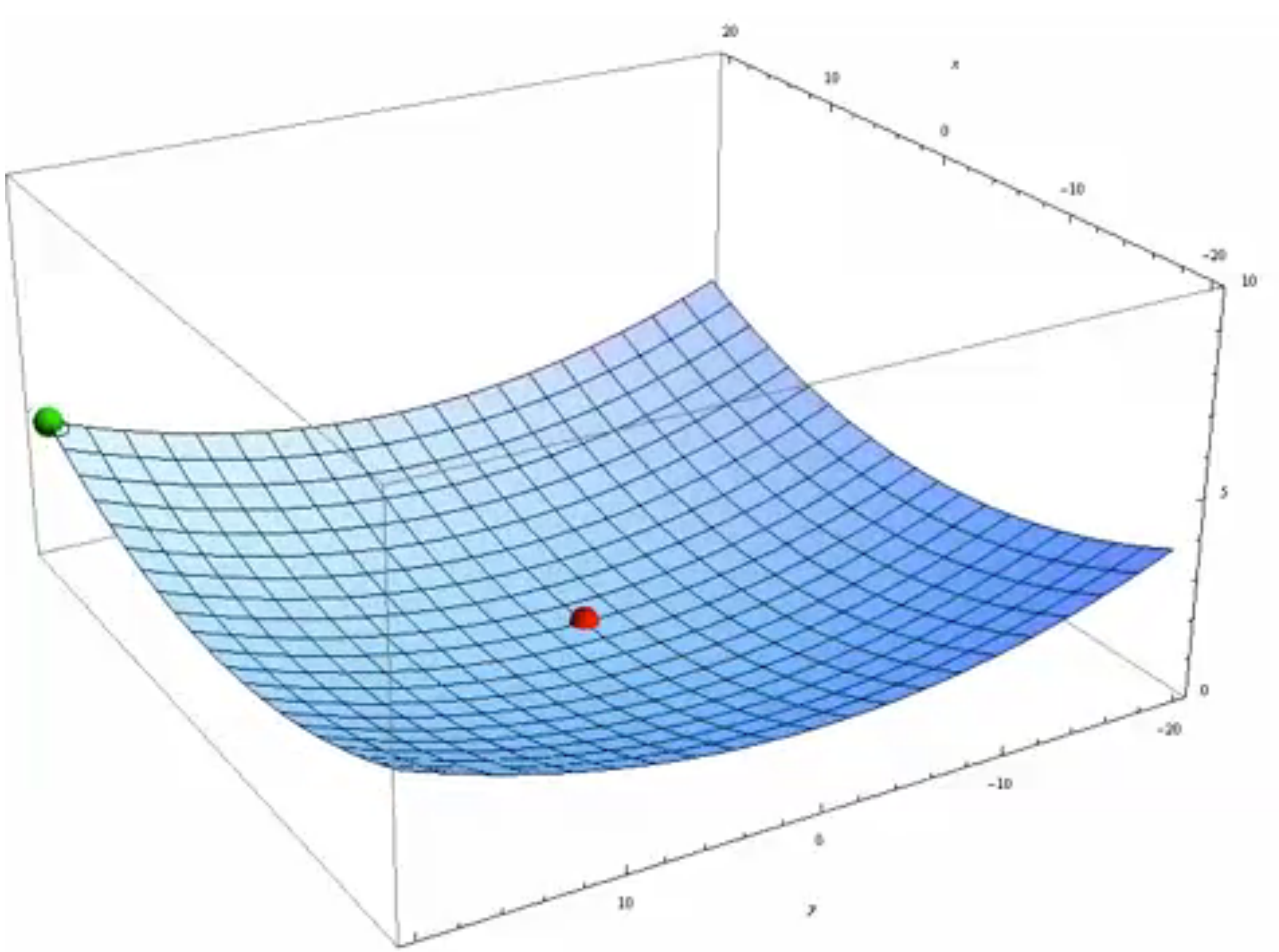
"Learning Rate"

Gradient-Descent Optimization

- Works for any function that can have a gradient estimated.
- Guaranteed to converge towards local-minima.
- Scales well to extremely large amounts of data.
- Notoriously slow (linear convergence).
- Often guess work associated tuning the learning rate.

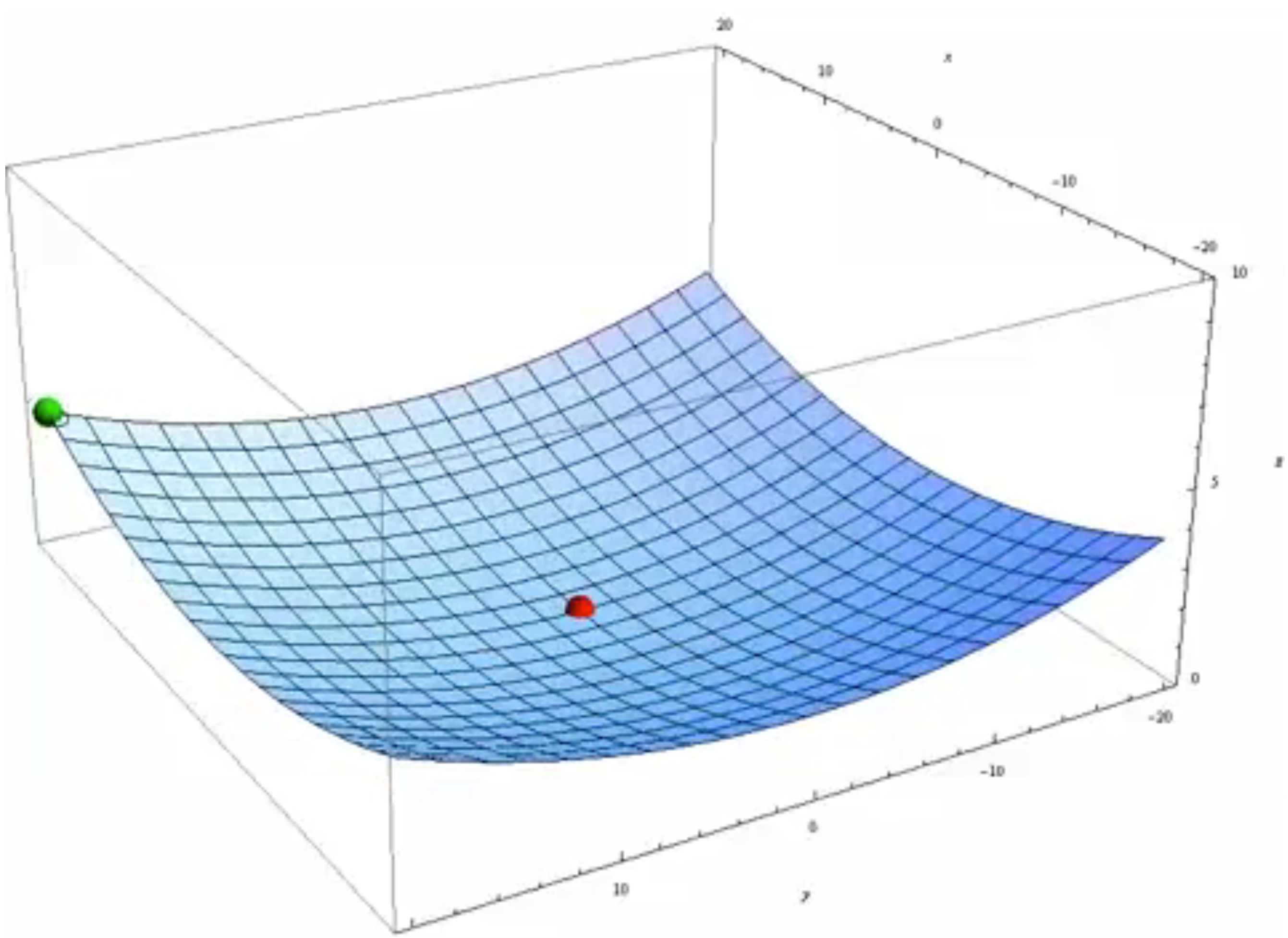
Optimizing Weights

$$\begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} + \eta \begin{bmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_K} \end{bmatrix}$$



Optimizing Weights

$$\begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} + \eta \begin{bmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_K} \end{bmatrix}$$



Optimizing Weights - Per Sample

- Objective nearly always summation over N samples,

$$f(\mathbf{w}) = \sum_{n=1}^{N} f_n(\mathbf{w})$$

- So one can update the weights per sample,

$$\mathbf{w} \rightarrow \mathbf{w} - \frac{\eta}{N} \frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}}$$

"Learning Rate"

Single Layer - Example

$$f_n(\mathbf{w}) = \frac{1}{2}||1 - t_n \cdot \mathbf{x}_n^T \mathbf{w}||_2^2$$

Single Layer - Example

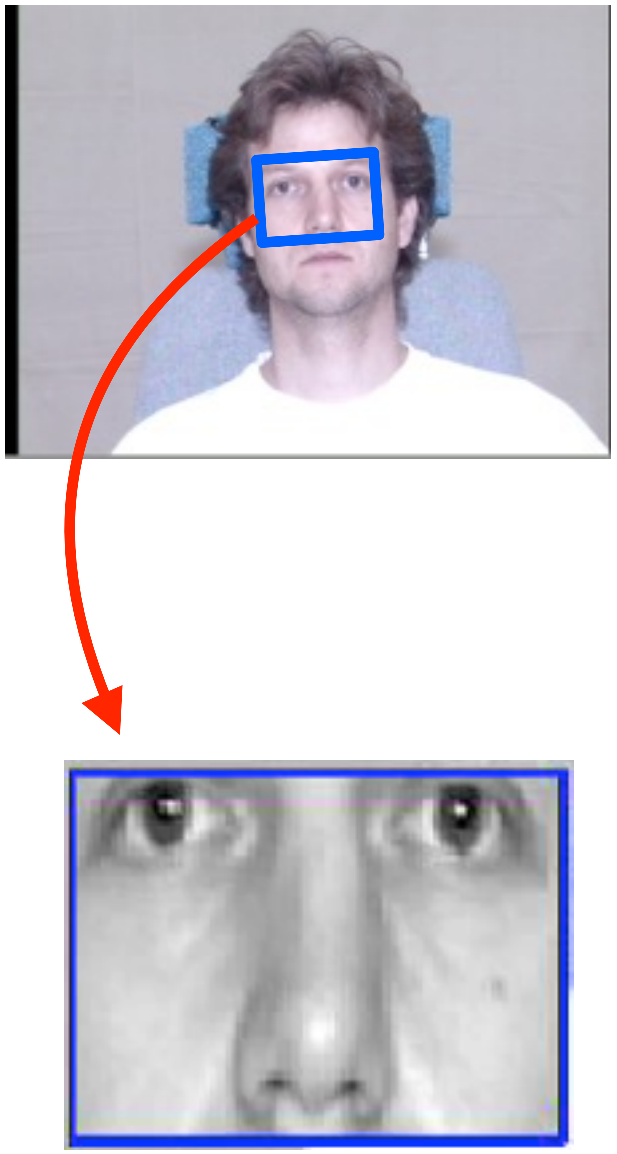
$$f_n(\mathbf{w}) = \frac{1}{2}||1 - t_n \cdot \mathbf{x}_n^T \mathbf{w}||_2^2$$

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}} = (\mathbf{x}_n^T \mathbf{w} - t_n)\mathbf{x}_n$$

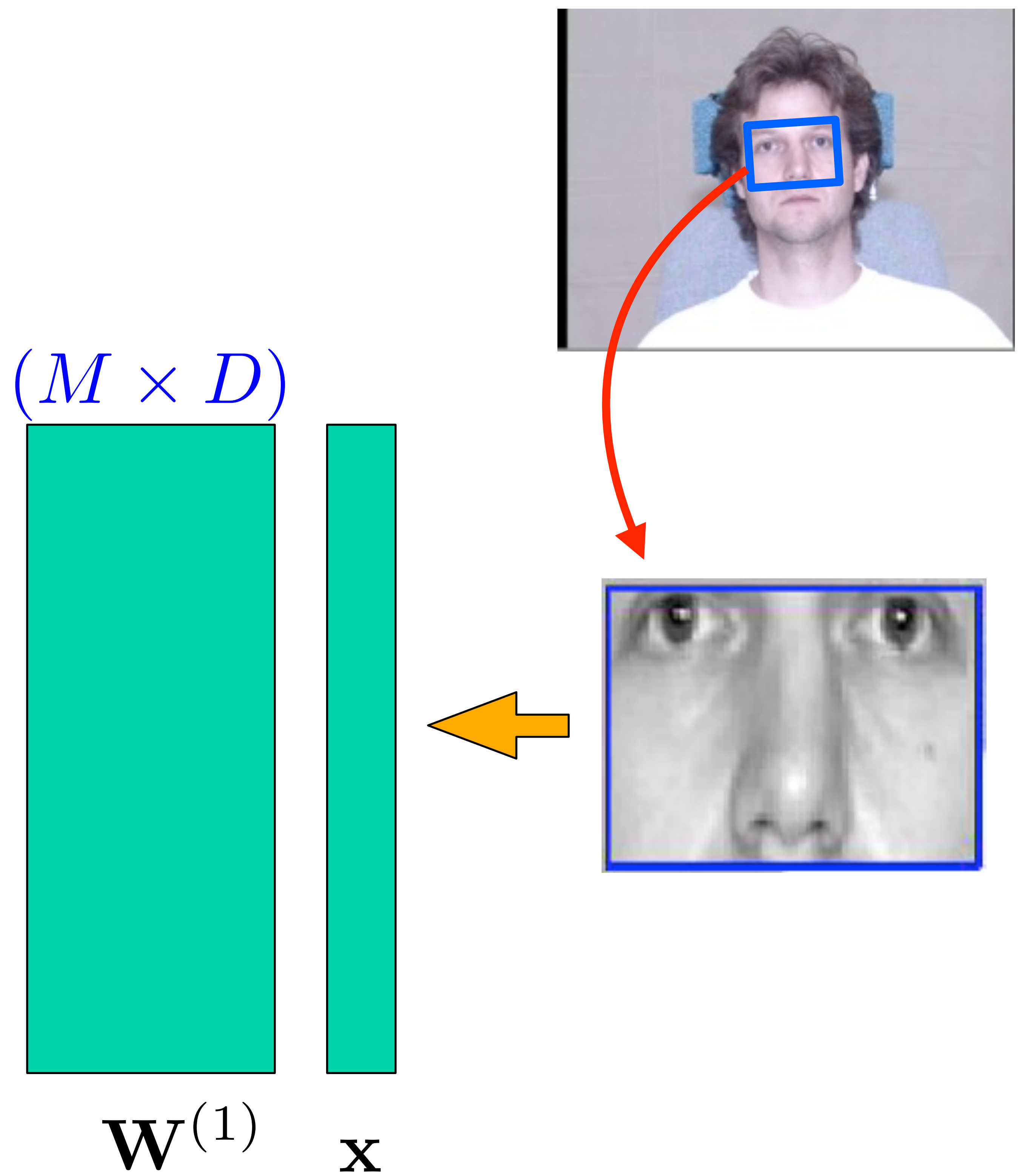
Today

- Single-Layer Perceptron
- Multi-Layer Perceptron
- Convolutional Neural Network

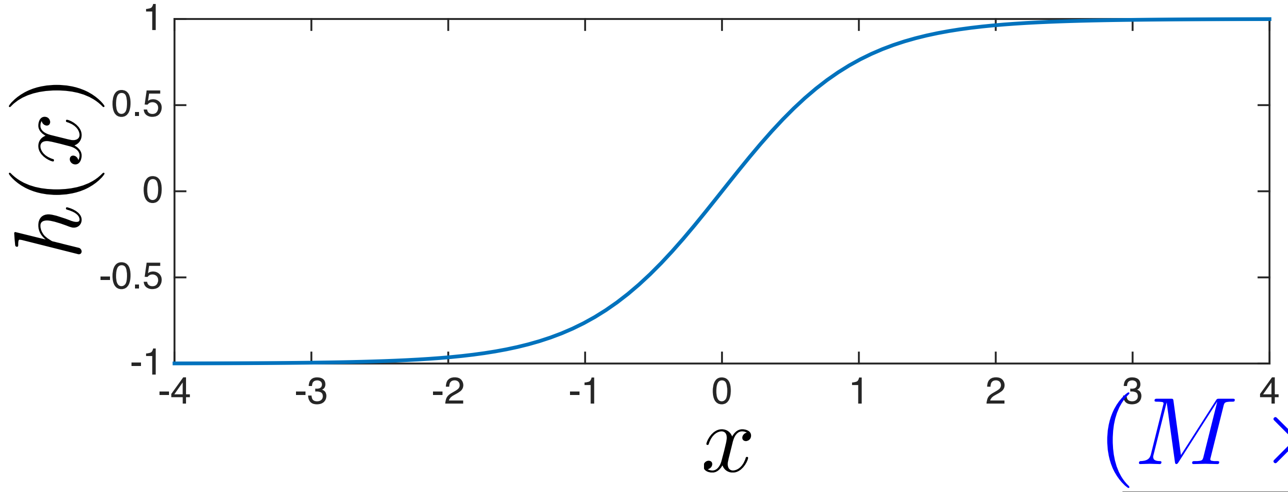
Multi-Layer Perceptron

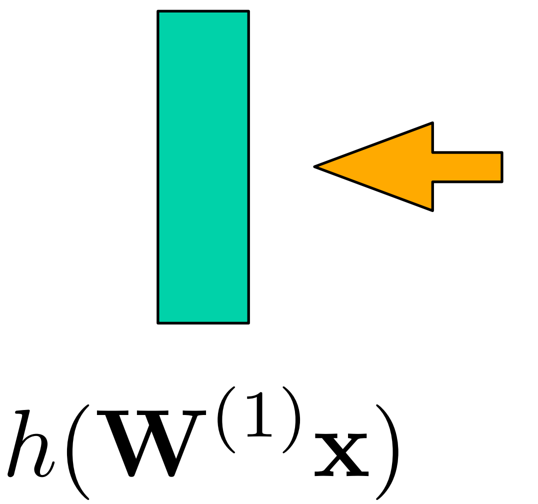


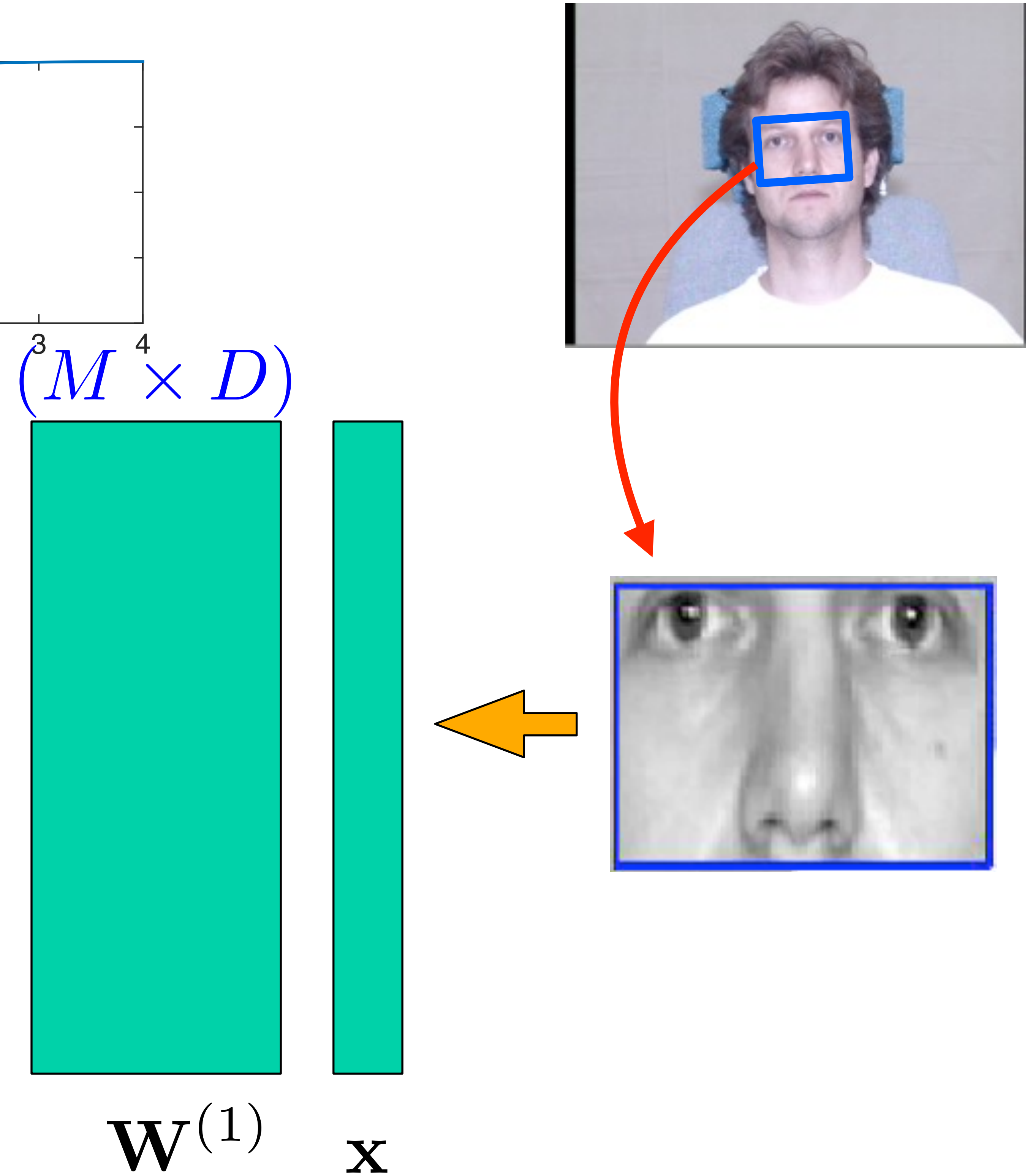
Multi-Layer Perceptron



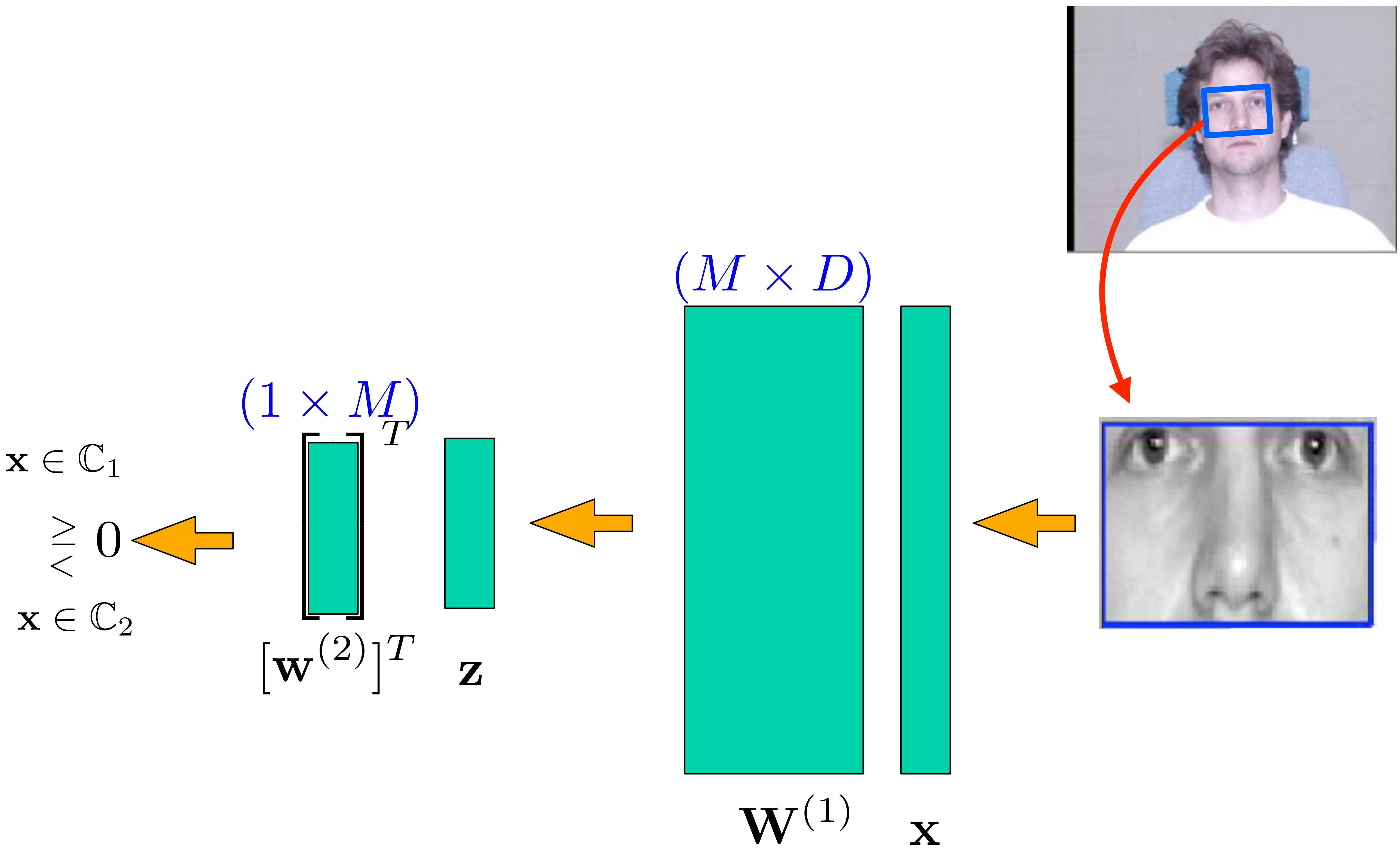
Multi-Layer Perceptron



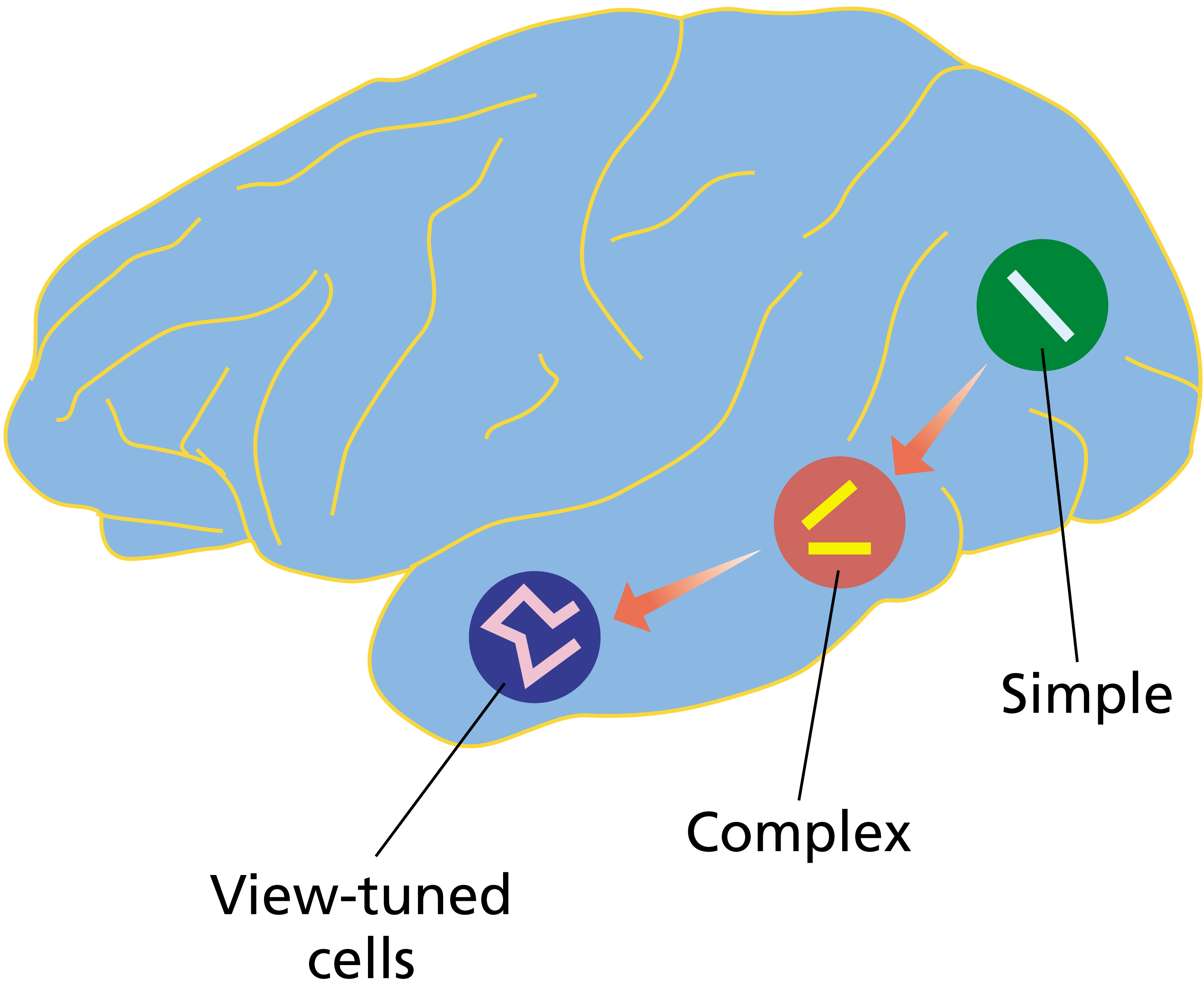




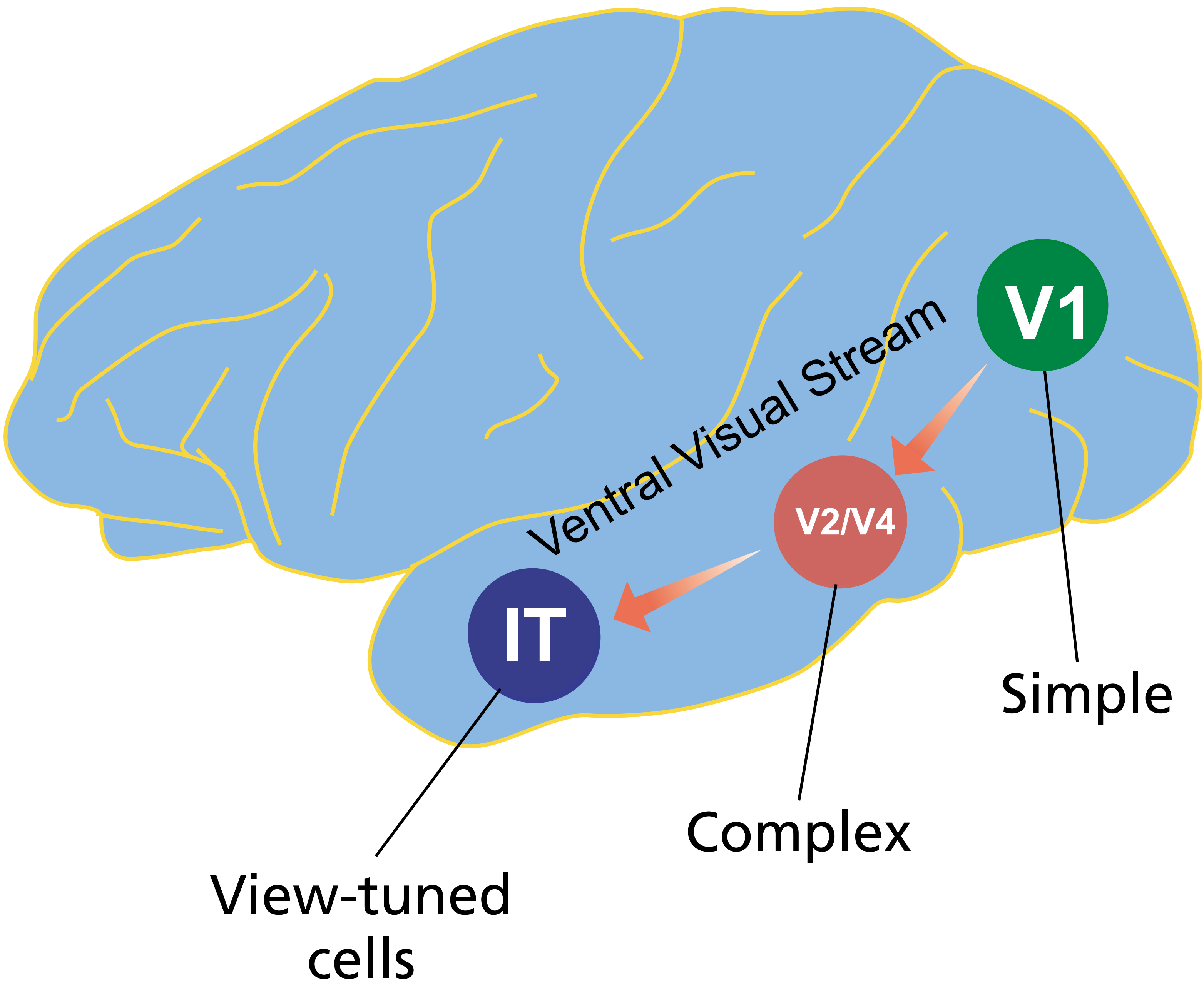
Multi-Layer Perceptron



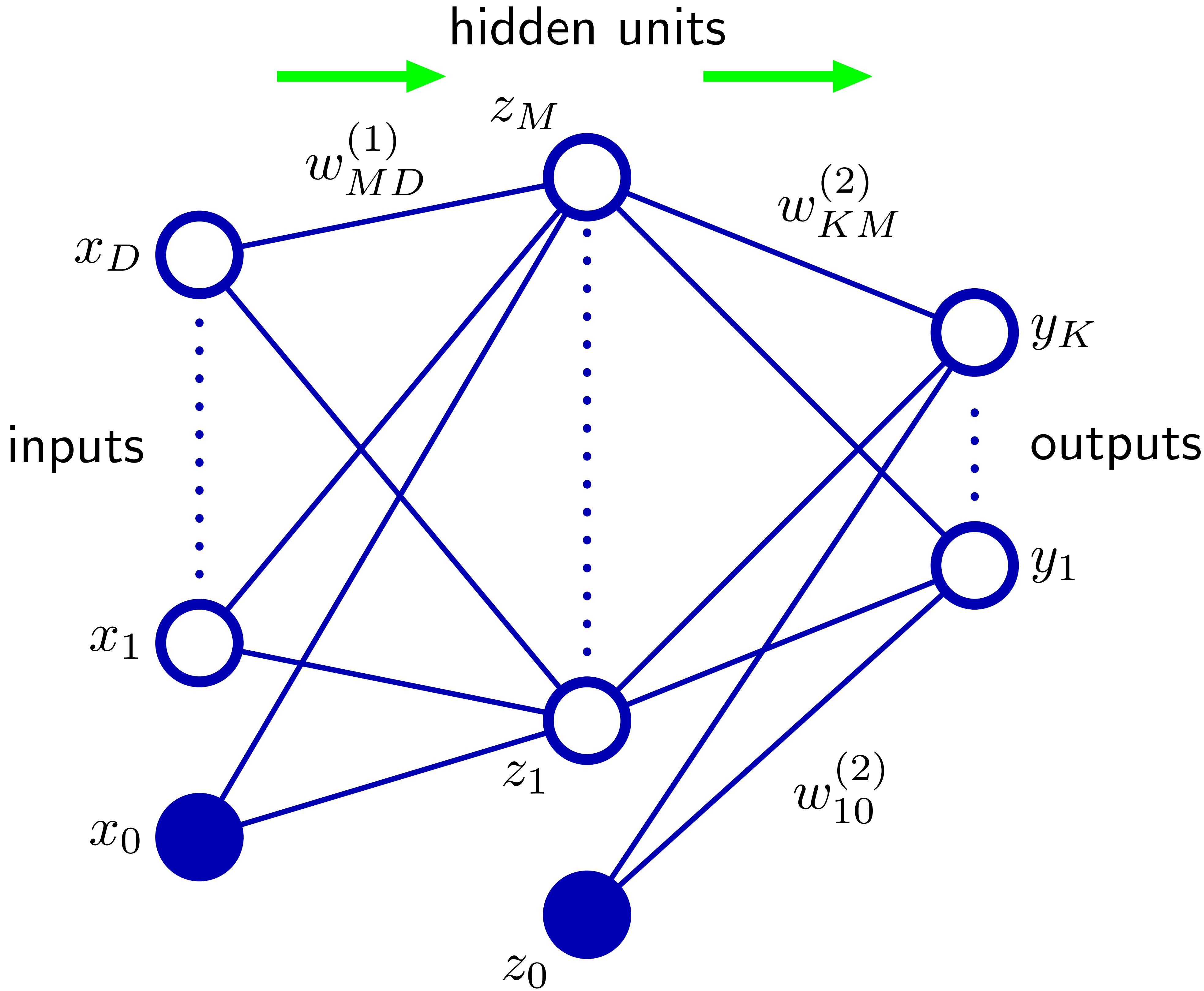
Reminder: Hierarchical Learning



Reminder: Hierarchical Learning



Multi-Layer Perceptron



Layer 1 - MLP

$$\mathbf{z} = \begin{bmatrix} z_1 \\ \vdots \\ z_M \end{bmatrix} \leftarrow \begin{bmatrix} h[\mathbf{x}^T \mathbf{w}_1^{(1)}] \\ \vdots \\ h[\mathbf{x}^T \mathbf{w}_M^{(1)}] \end{bmatrix}$$

$h() =$ non-linear function

$[\mathbf{w}_1^{(1)}, \ldots, \mathbf{w}_M^{(1)}] =$ 1st layer's $D \times M$ weights

$\mathbf{x} = D \times 1$ raw input

Layer 2 - MLP

[65,09,67,.......,78,66,76,215] ^{T}

$\mathbf{x} \in \mathbb{R}^D$

$\mathbf{z} \in \mathbb{R}^M$

$$\mathbf{z}^T \mathbf{w}^{(2)} \underset{\mathbf{z} \in \mathbb{C}_2}{\overset{\mathbf{z} \in \mathbb{C}_1}{\gtrless}} 0$$

$\mathbf{z} = M \times 1$ output of layer 1

$\mathbf{w}^{(2)} = $ 2nd layer's $M \times 1$ weight vector

Obvious Questions?

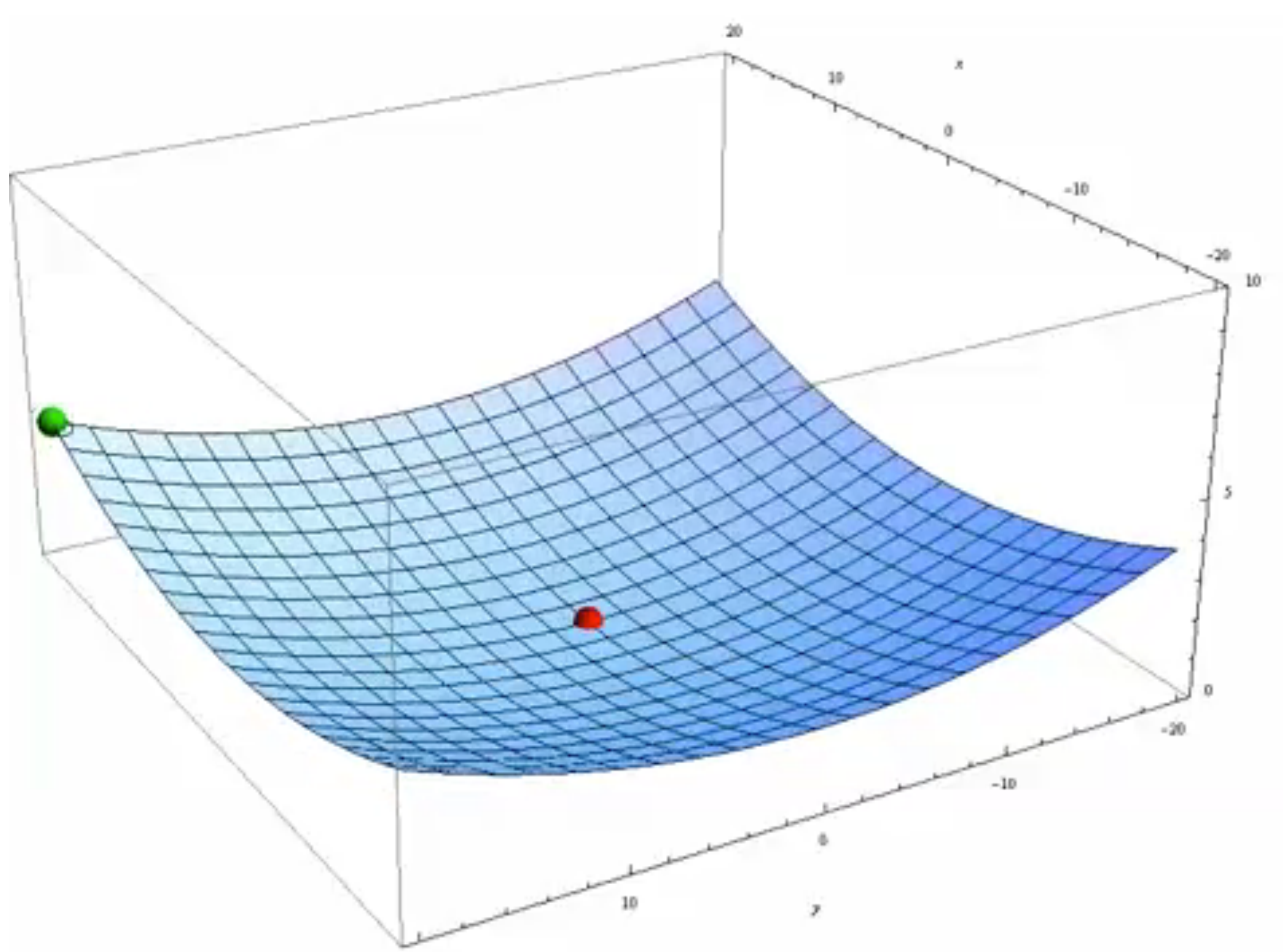
- How many layers?
- Is the solution globally optimal?
- What non-linearity should you use?
- What learning rate?
- How to should I estimate my gradients?

Obvious Questions?

- How many layers?
- Is the solution globally optimal?
- What non-linearity should you use?
- What learning rate?
- How to should I estimate my gradients?

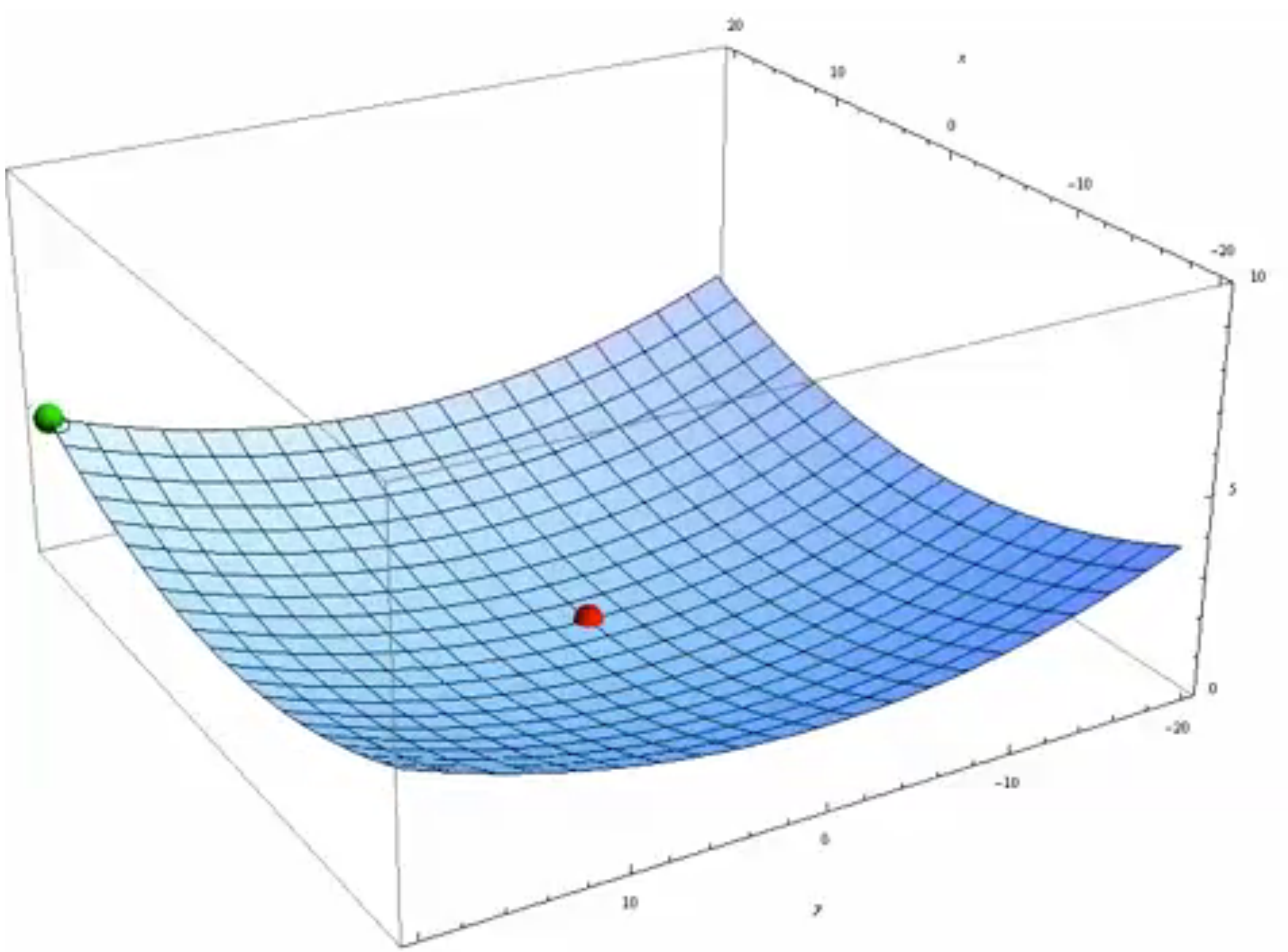
Back-Propagation

$$\begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} + \eta \begin{bmatrix} \dfrac{\partial f(\mathbf{w})}{\partial w_1} \\ \vdots \\ \dfrac{\partial f(\mathbf{w})}{\partial w_K} \end{bmatrix}$$



Back-Propagation

$$\begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} \leftarrow \begin{bmatrix} w_1 \\ \vdots \\ w_K \end{bmatrix} + \eta \begin{bmatrix} \frac{\partial f(\mathbf{w})}{\partial w_1} \\ \vdots \\ \frac{\partial f(\mathbf{w})}{\partial w_K} \end{bmatrix}$$



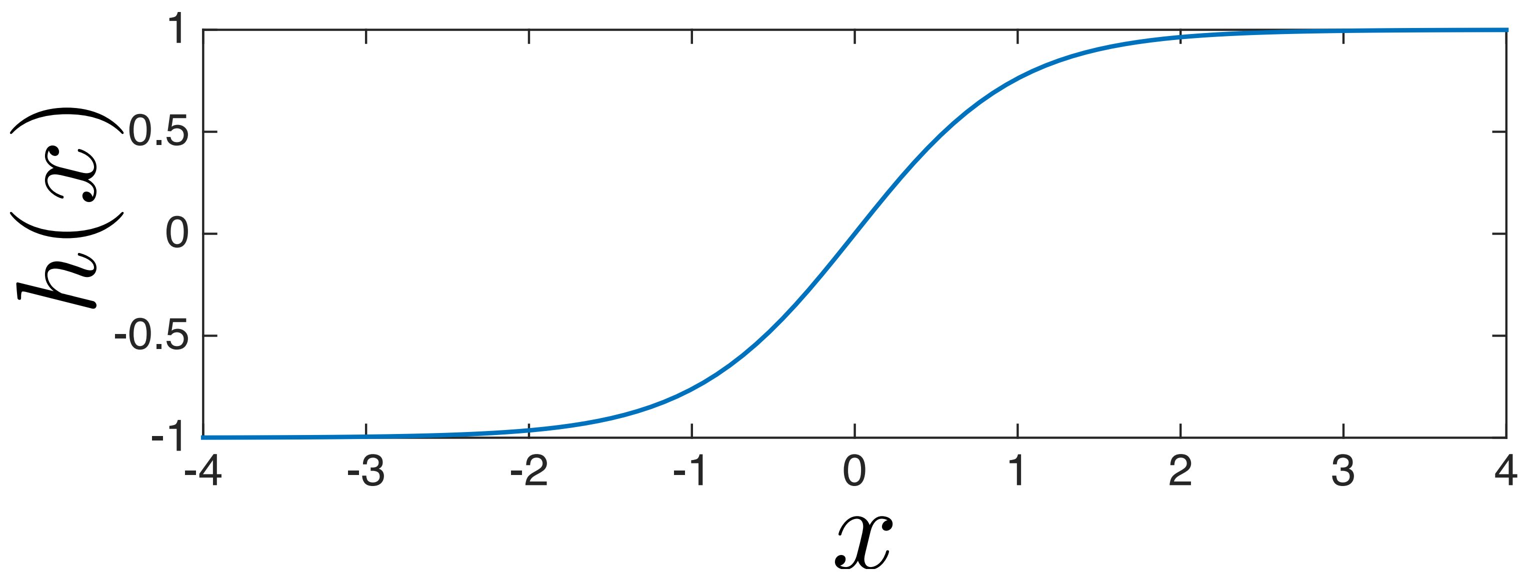
Two Layer - Example

$$f_n(\mathbf{w}) = \frac{1}{2}||1 - t_n \cdot \mathbf{z}_n^T \mathbf{w}^{(2)}||_2^2$$

$$\text{s.t. } \mathbf{z}_n = [z_1, \ldots, z_M]^T$$

$$z_m = h(\mathbf{x}_n^T \mathbf{w}_m^{(1)})$$

$$\text{where } h(x) = \frac{1}{1 + \exp(-x)}, \quad \frac{\partial h(x)}{\partial x} = 1 - h(x)^2$$



Two Layer - Example

From previous example we know for layer 2,

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = (\mathbf{z}_n^T \mathbf{w}^{(2)} - t_n)\mathbf{z}_n$$

Two Layer - Example

Using the chain rule we can determine that,

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

Two Layer - Example

Using the chain rule we can determine that,

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

$$\frac{\partial f_n(\mathbf{w})}{\partial z_m} = (\mathbf{z}_n^T \mathbf{w}^{(2)} - t_n) w_m^{(2)}$$

Two Layer - Example

Using the chain rule we can determine that,

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

$$\frac{\partial f_n(\mathbf{w})}{\partial z_m} = (\mathbf{z}_n^T \mathbf{w}^{(2)} - t_n) w_m^{(2)}$$

$$\frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}} = [1 - h(\mathbf{x}_n^T \mathbf{w}_m^{(1)})^2] \mathbf{x}_n$$

Back Propagation

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = (\mathbf{z}_n^T \mathbf{w}^{(2)} - t_n)\mathbf{z}_n$$

Back Propagation

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = \delta_n \mathbf{z}_n$$

Back Propagation

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = \delta_n \mathbf{z}_n$$

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

$$\frac{\partial f_n(\mathbf{w})}{\partial z_m} = (\mathbf{z}_n^T \mathbf{w}^{(2)} - t_n) w_m^{(2)}$$

Back Propagation

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = \delta_n \mathbf{z}_n$$

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

$$\frac{\partial f_n(\mathbf{w})}{\partial z_m} = \delta_n w_m^{(2)}$$

Back Propagation

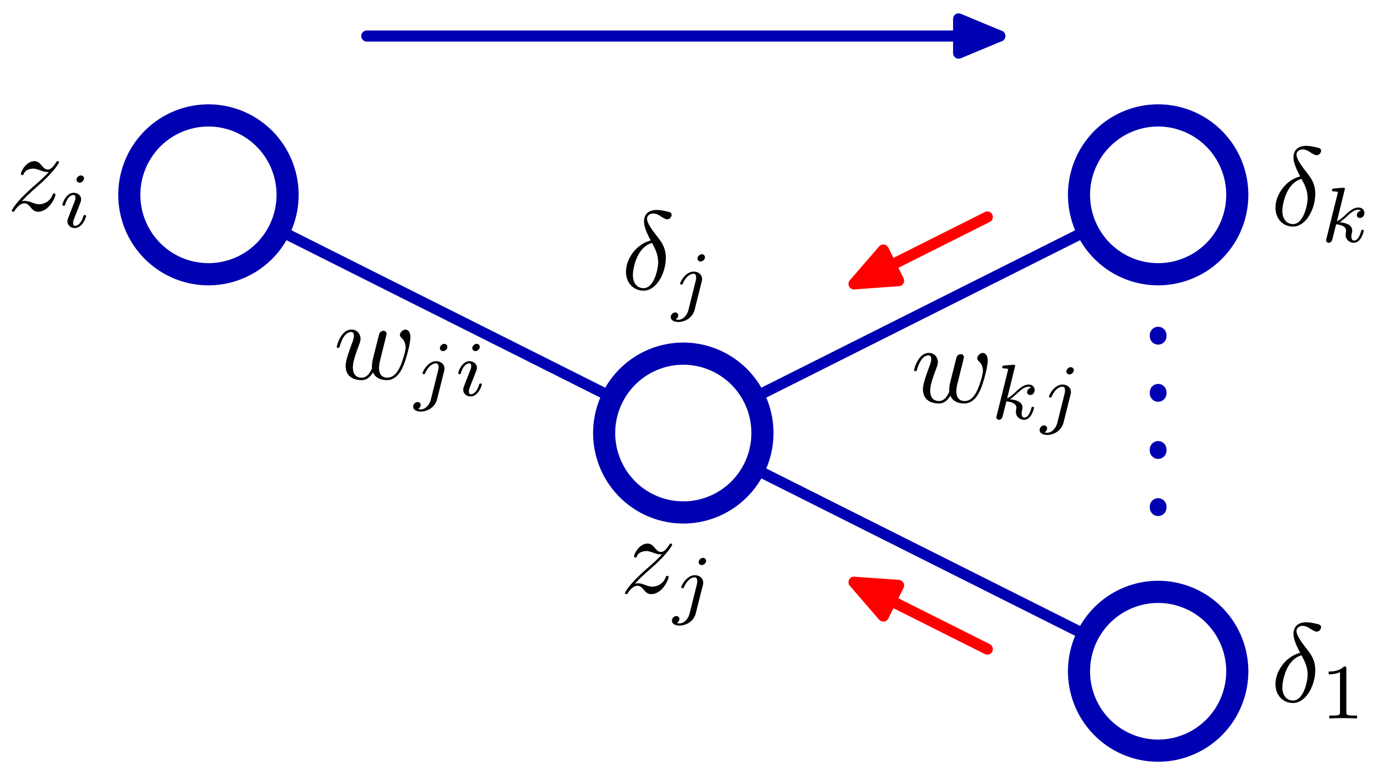
$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}^{(2)}} = \delta_n \mathbf{z}_n$$

$$\frac{\partial f_n(\mathbf{w})}{\partial \mathbf{w}_m^{(1)}} = \frac{\partial f_n(\mathbf{w})}{\partial z_m} \frac{\partial z_m}{\partial \mathbf{w}_m^{(1)}}$$

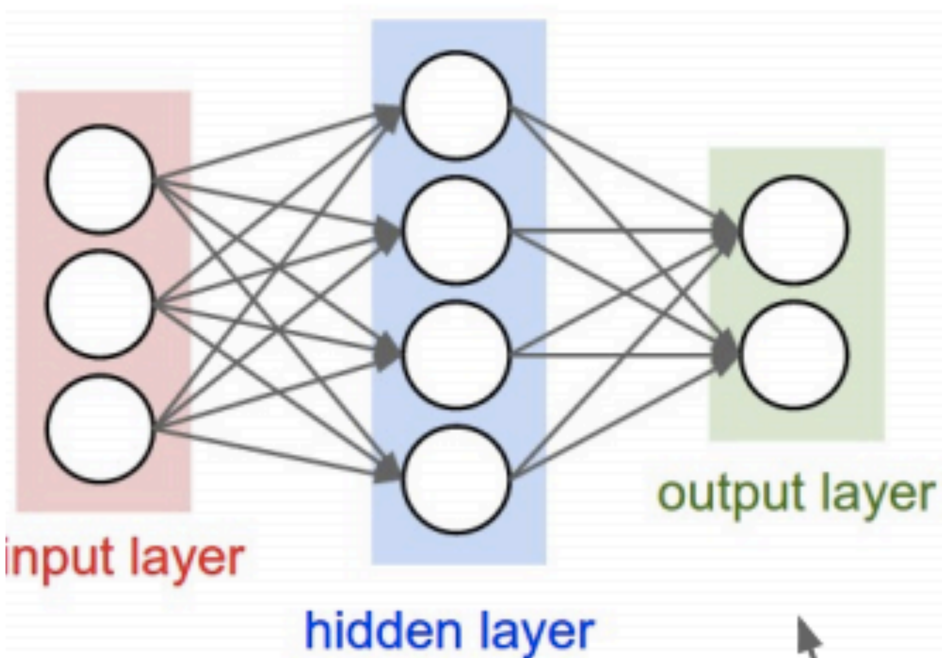
$$\frac{\partial f_n(\mathbf{w})}{\partial z_m} = \delta_n w_m^{(2)}$$

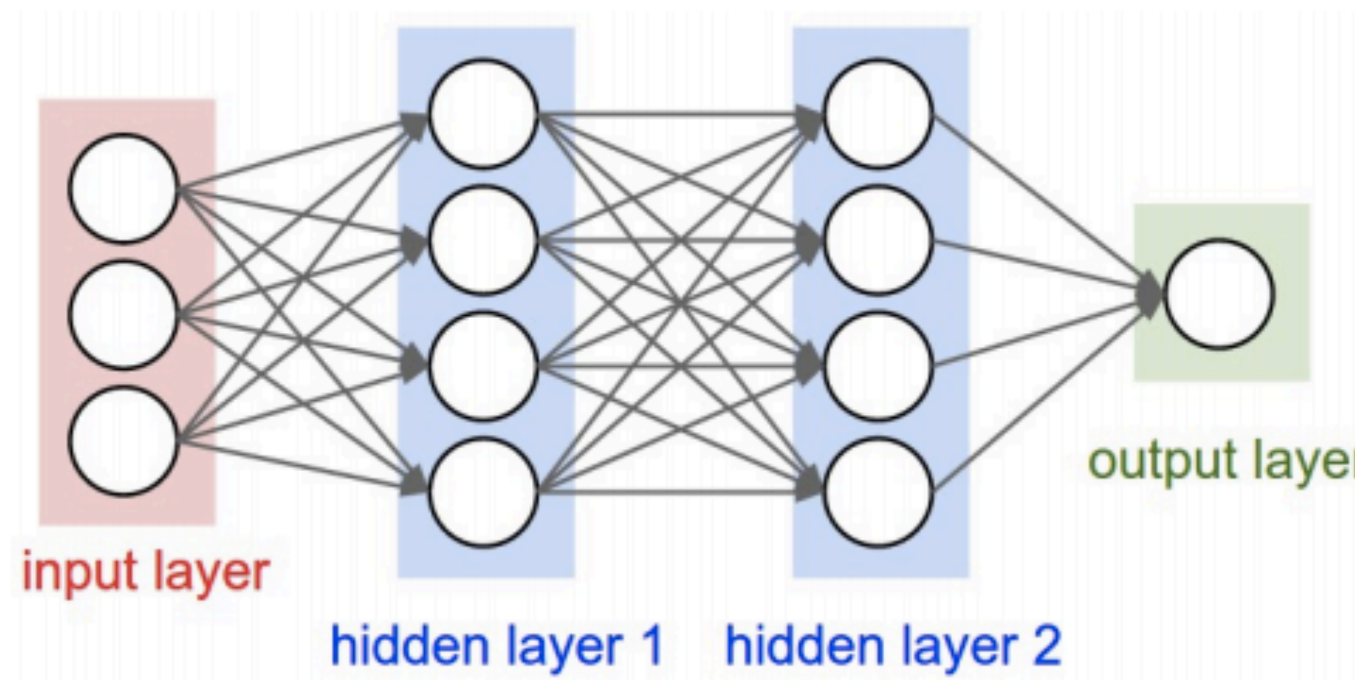
Back Propagation

- Back propagation refers to the property that components of gradients found at higher layers, can be re-used at lower layers.



Multiple Layers





“2-layer neural net,” or
“1-hidden-layer neural net”

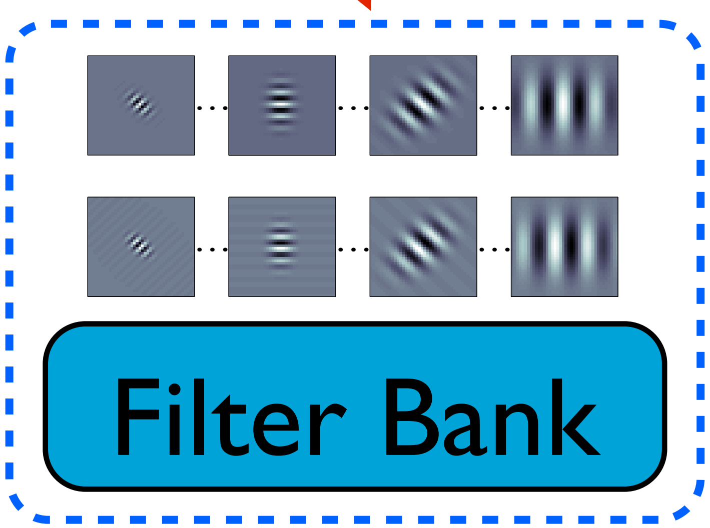
“Fully-connected” layers

“3-layer neural net,” or
“2-hidden-layer neural net”

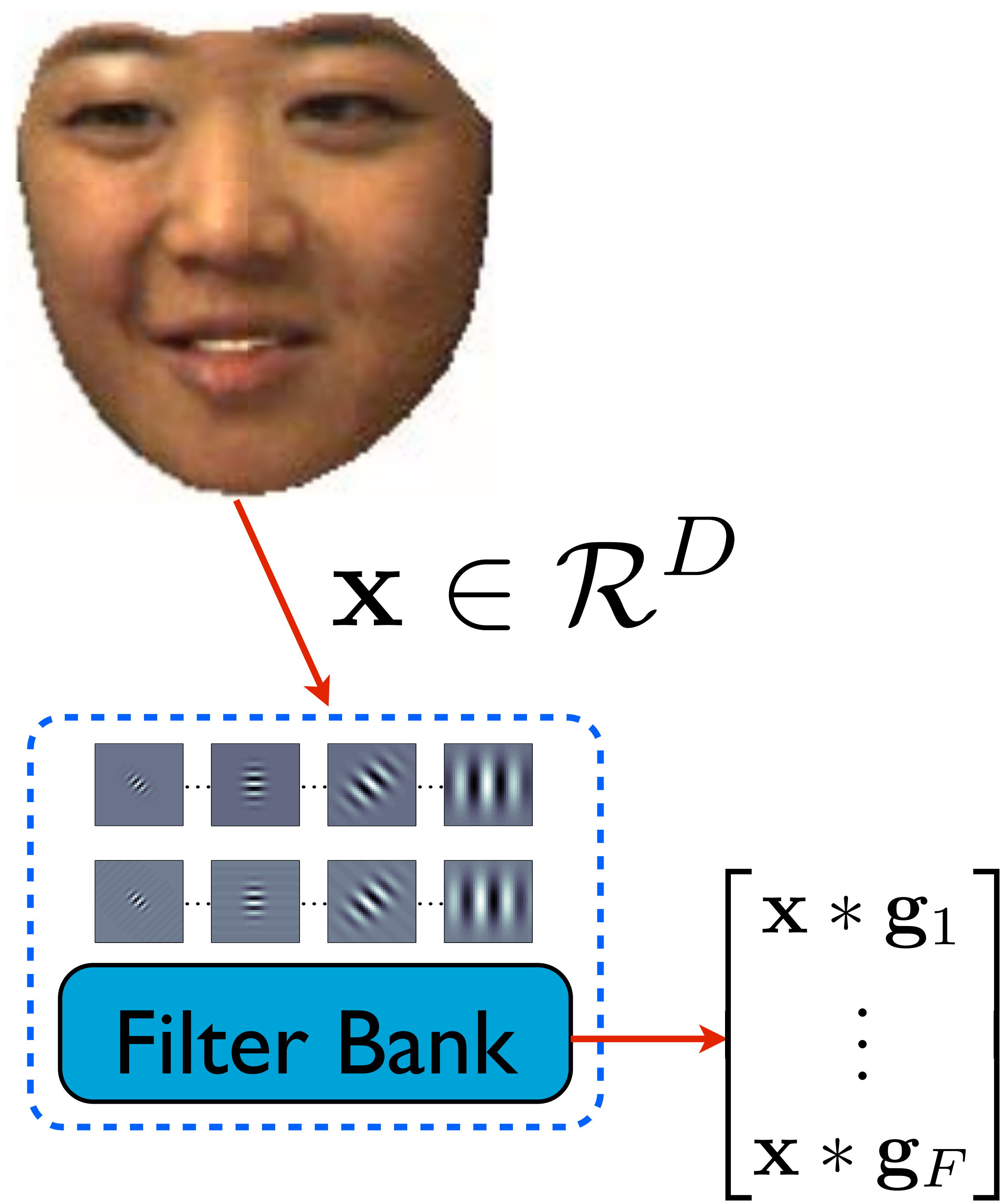
Today

- Single-Layer Perceptron
- Multi-Layer Perceptron
- Convolutional Neural Network

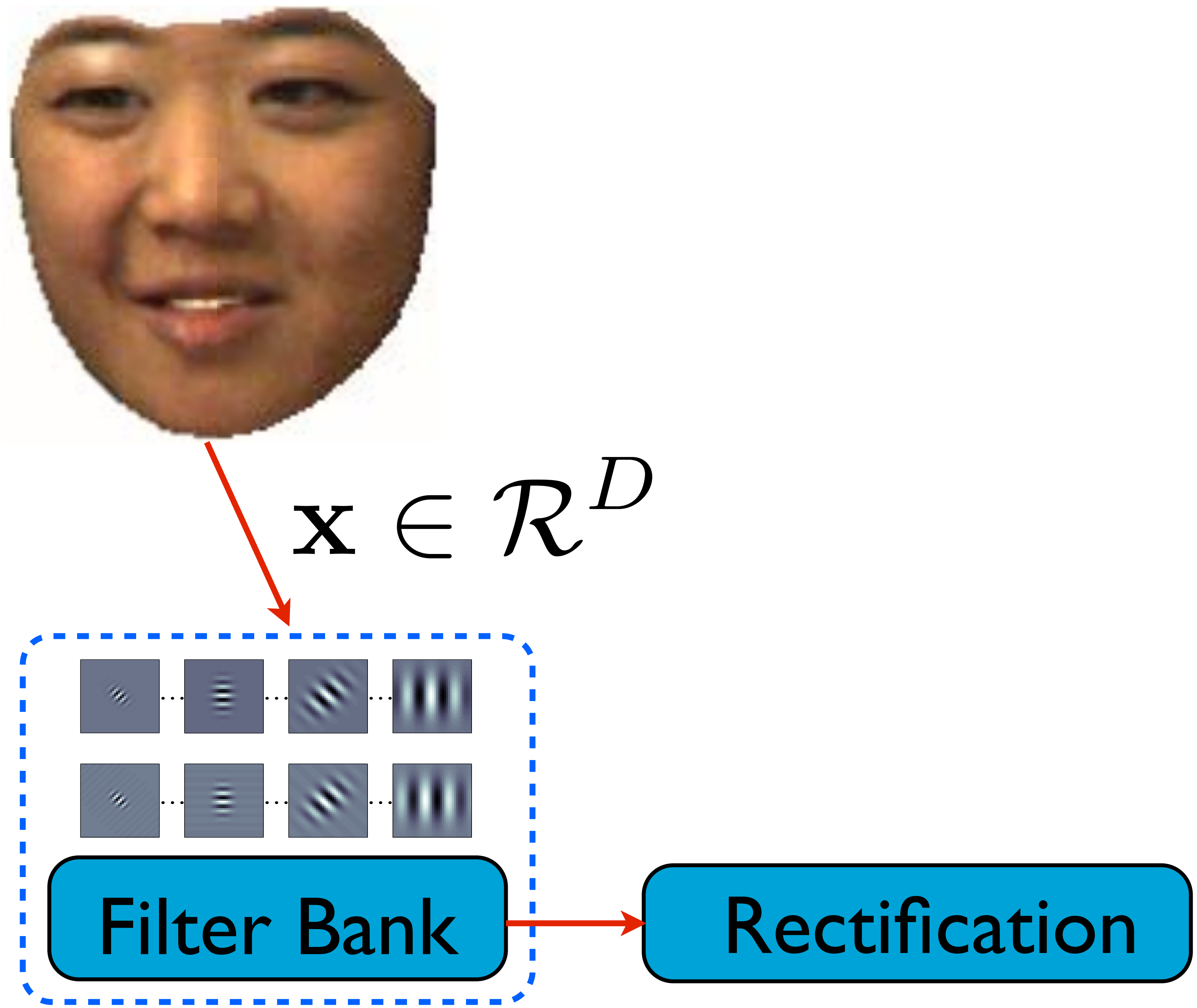
$$\mathbf{x} \in \mathcal{R}^D$$



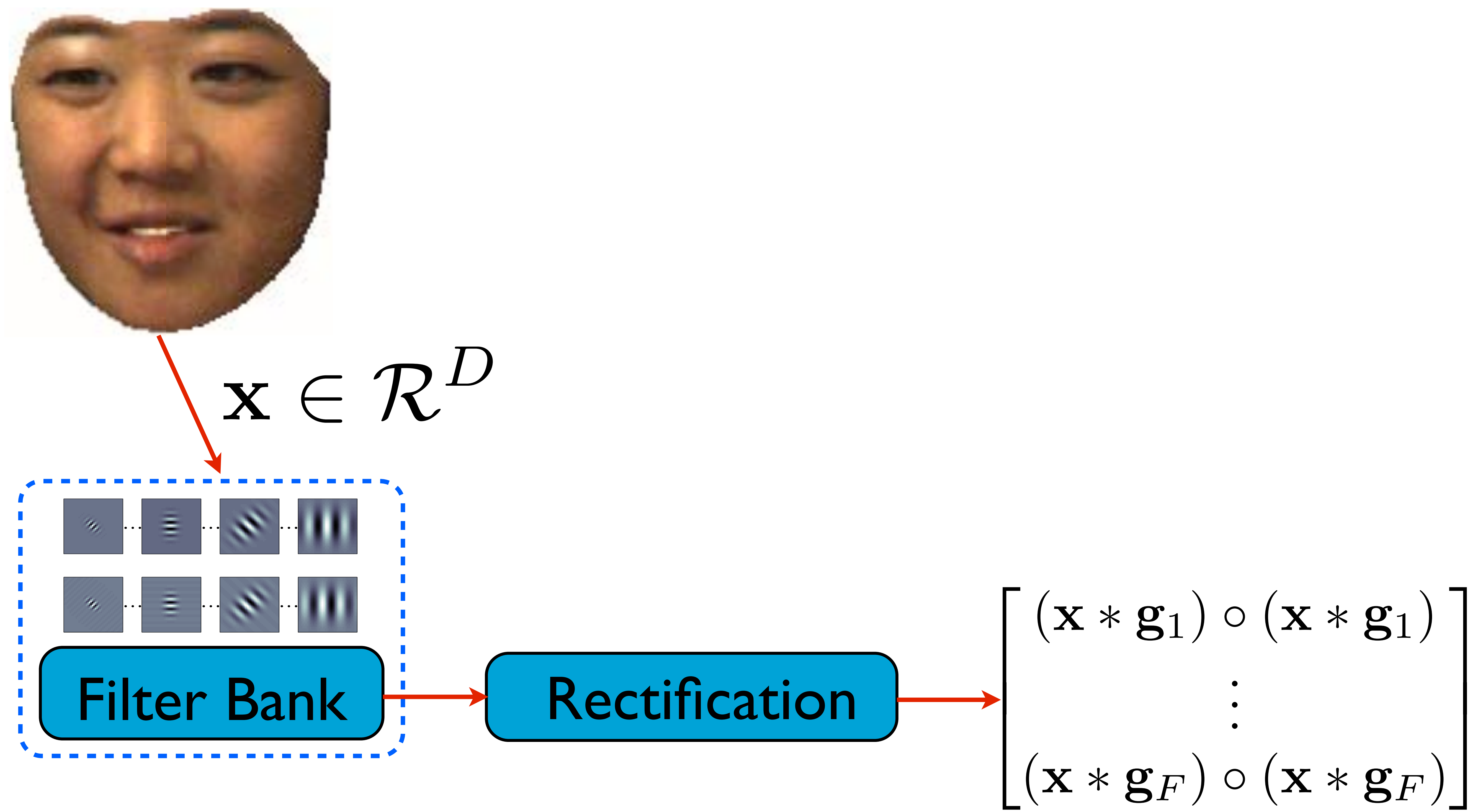
Hubel & Wiesel 1962
Serre & Poggio 2007
Lowe 1999
Pinto et al. 2008



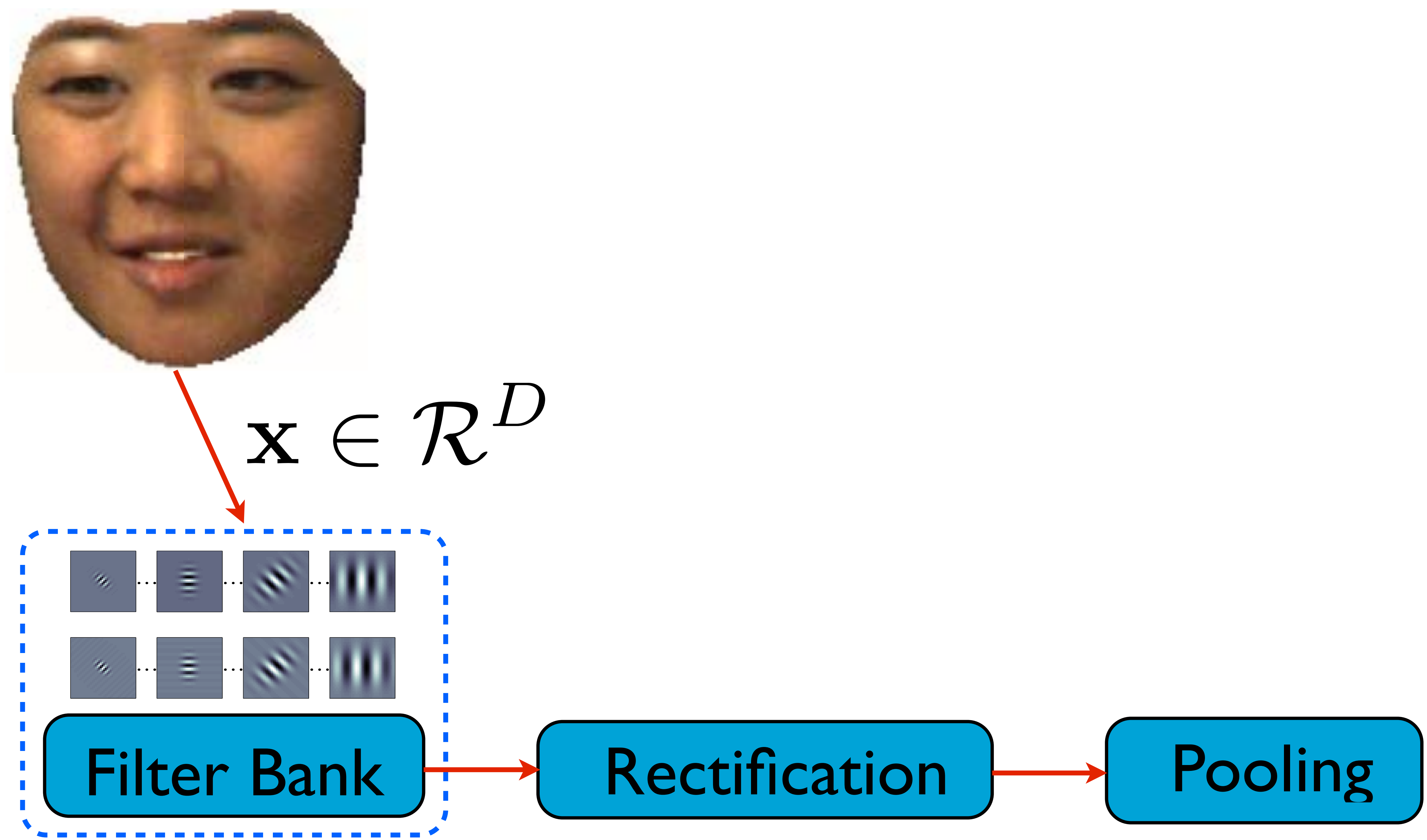
Hubel & Wiesel 1962
Lowe 1999
Serre & Poggio 2007
Pinto et al. 2008



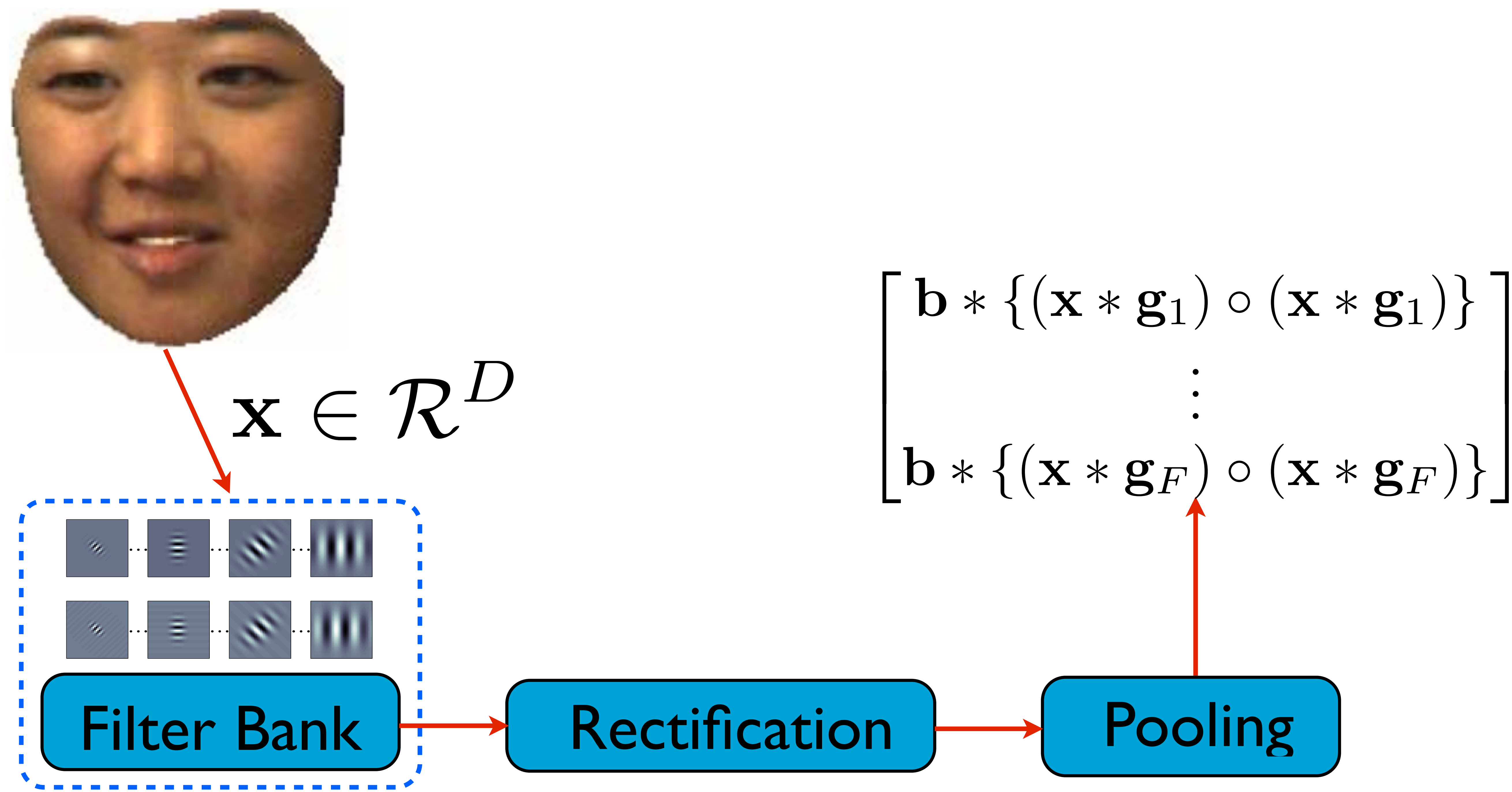
Hubel & Wiesel 1962 Lowe 1999
Serre & Poggio 2007 Pinto et al. 2008



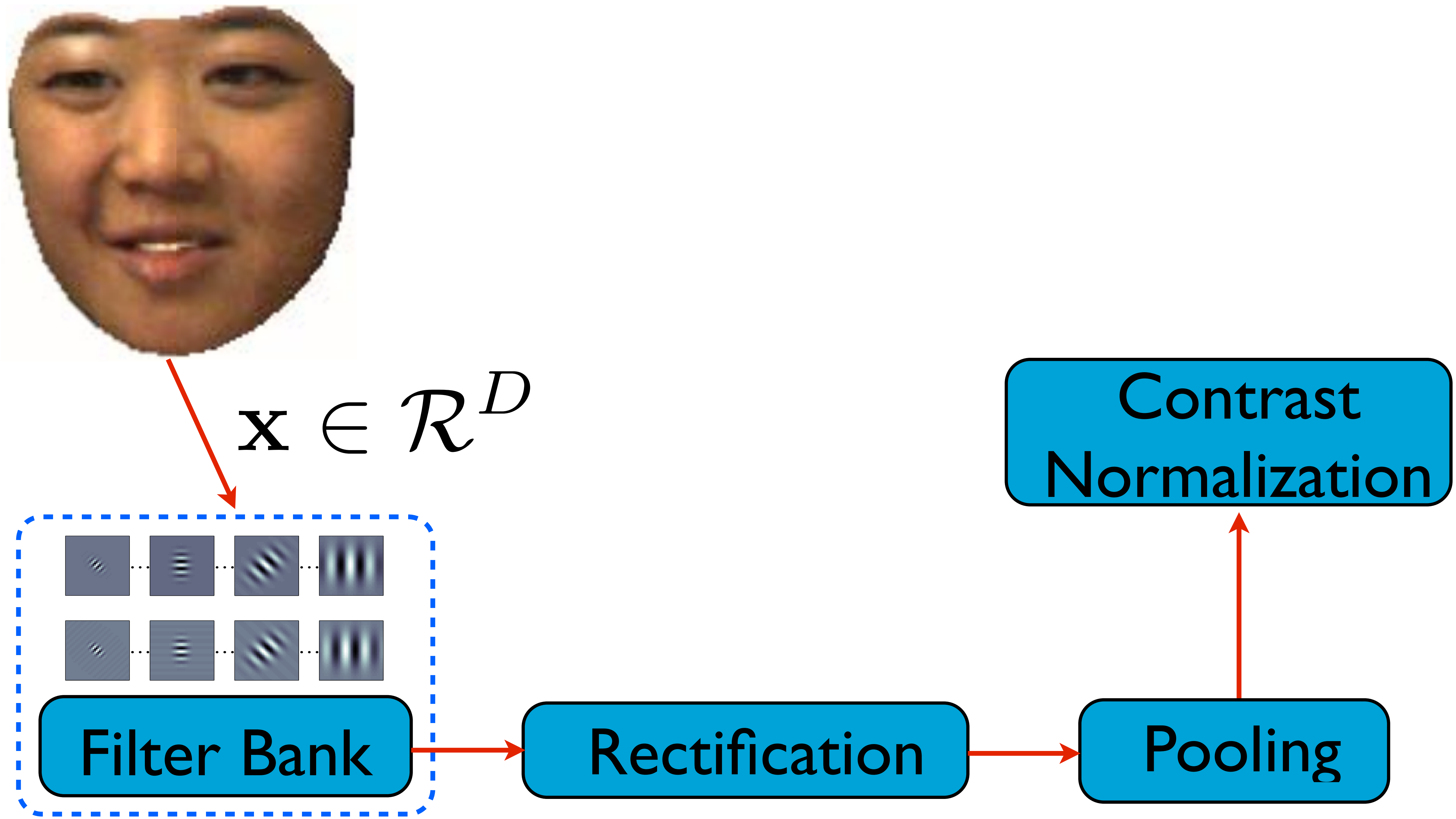
Hubel & Wiesel 1962 Lowe 1999
Serre & Poggio 2007 Pinto et al. 2008



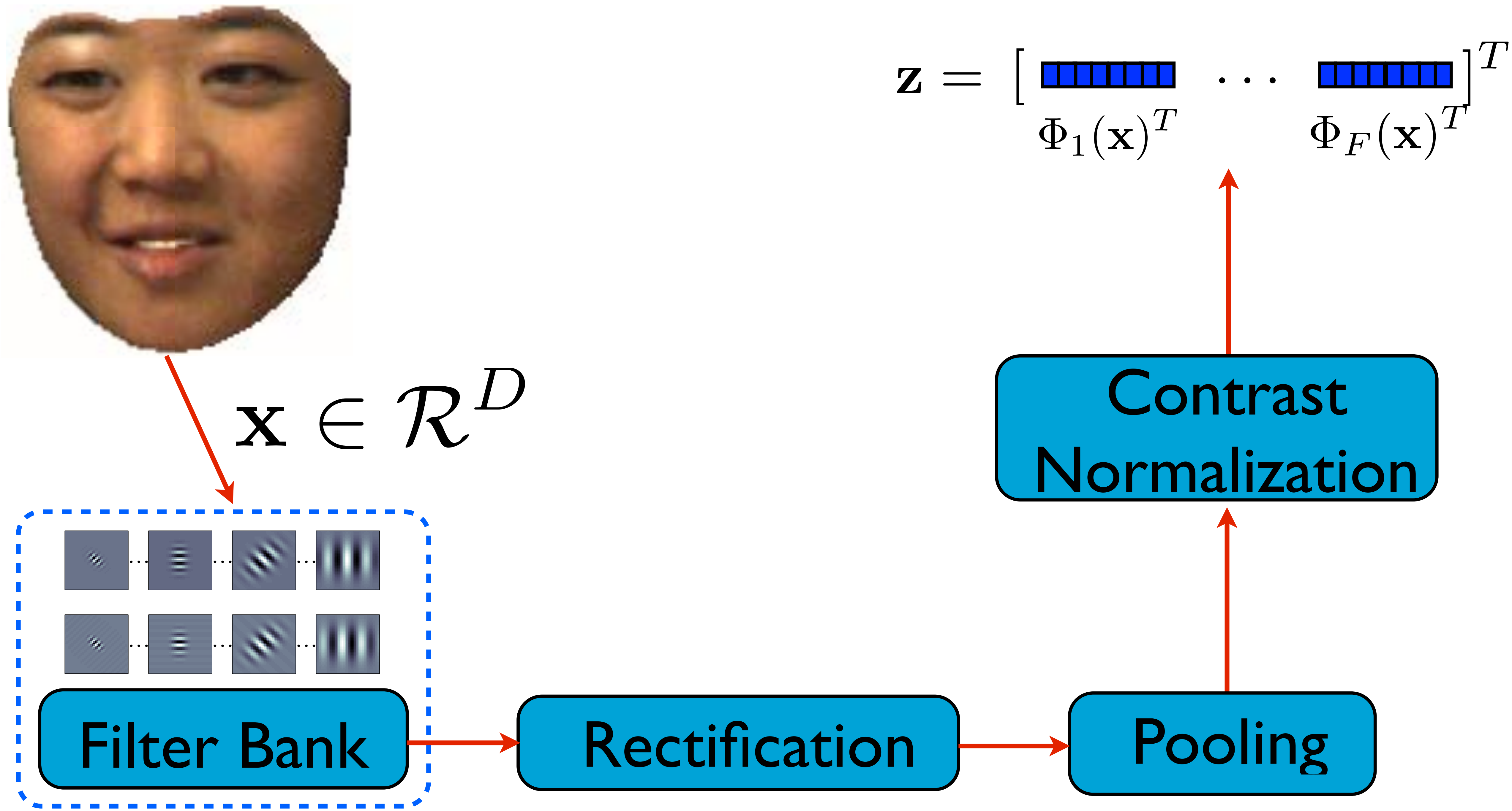
Hubel & Wiesel 1962
Serre & Poggio 2007
Lowe 1999
Pinto et al. 2008



Hubel & Wiesel 1962
Serre & Poggio 2007
Lowe 1999
Pinto et al. 2008



Hubel & Wiesel 1962
Serre & Poggio 2007
Lowe 1999
Pinto et al. 2008

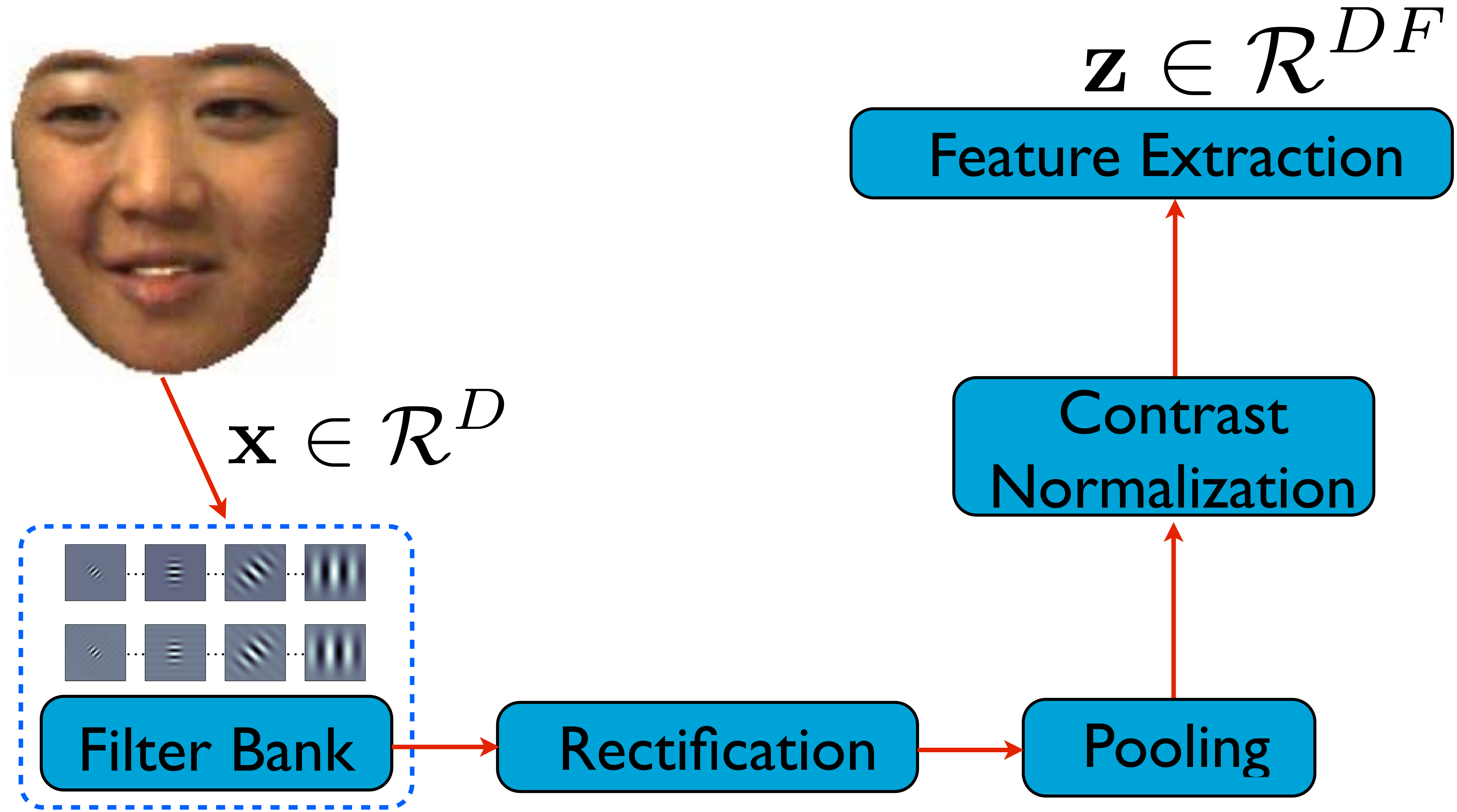


Hubel & Wiesel 1962

Serre & Poggio 2007

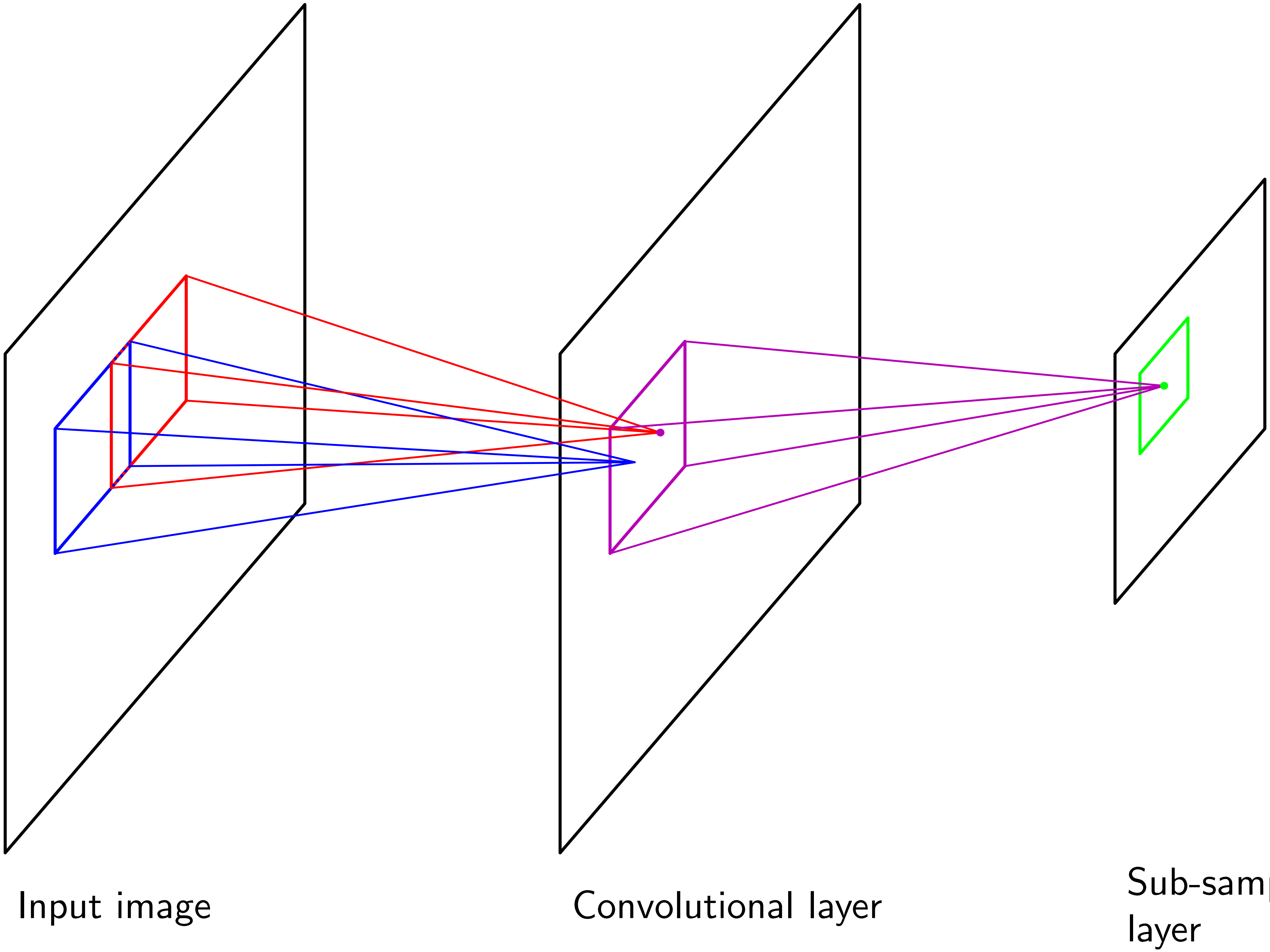
Lowe 1999

Pinto et al. 2008



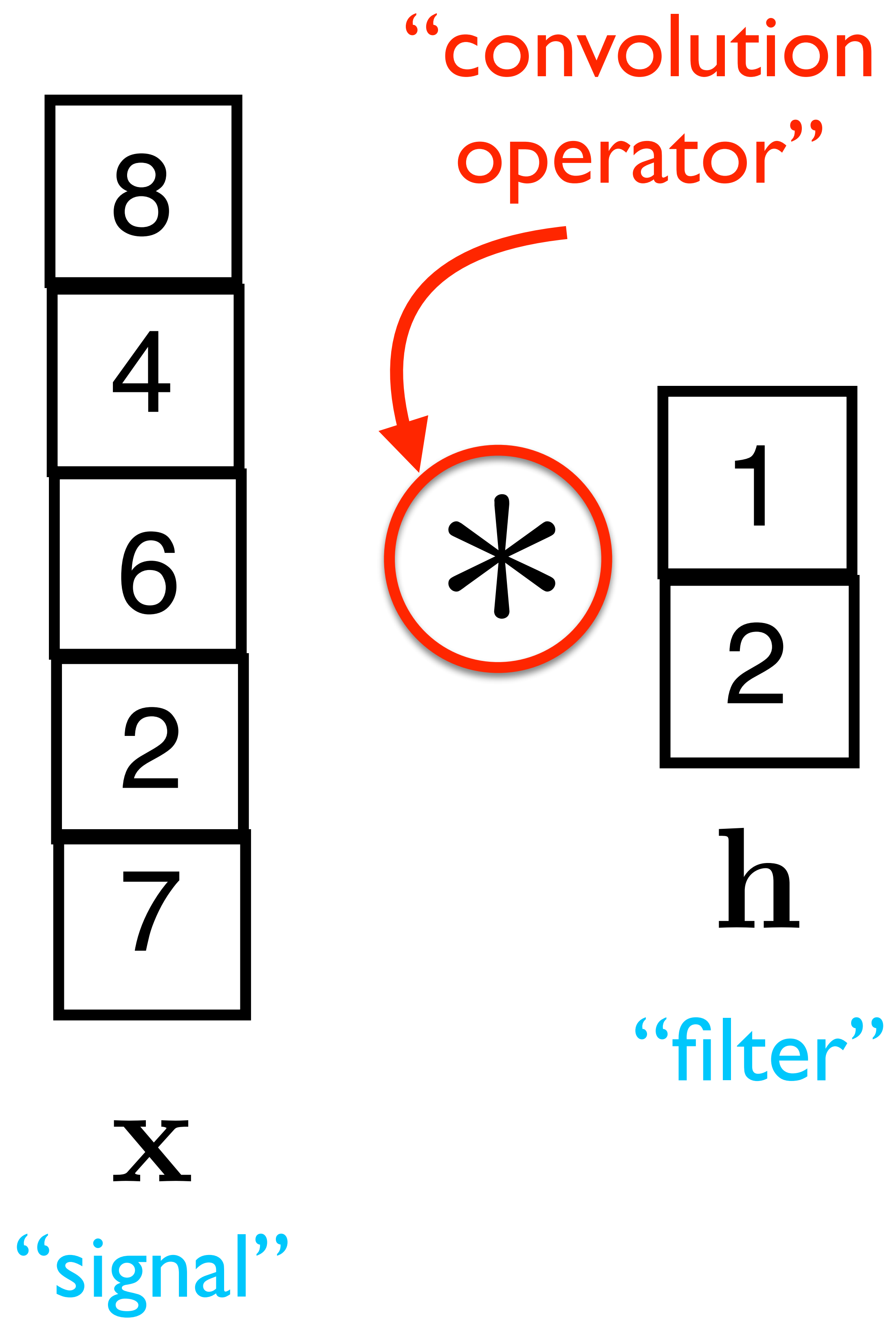
Hubel & Wiesel 1962 Lowe 1999
Serre & Poggio 2007 Pinto et al. 2008

Convolutional Neural Network



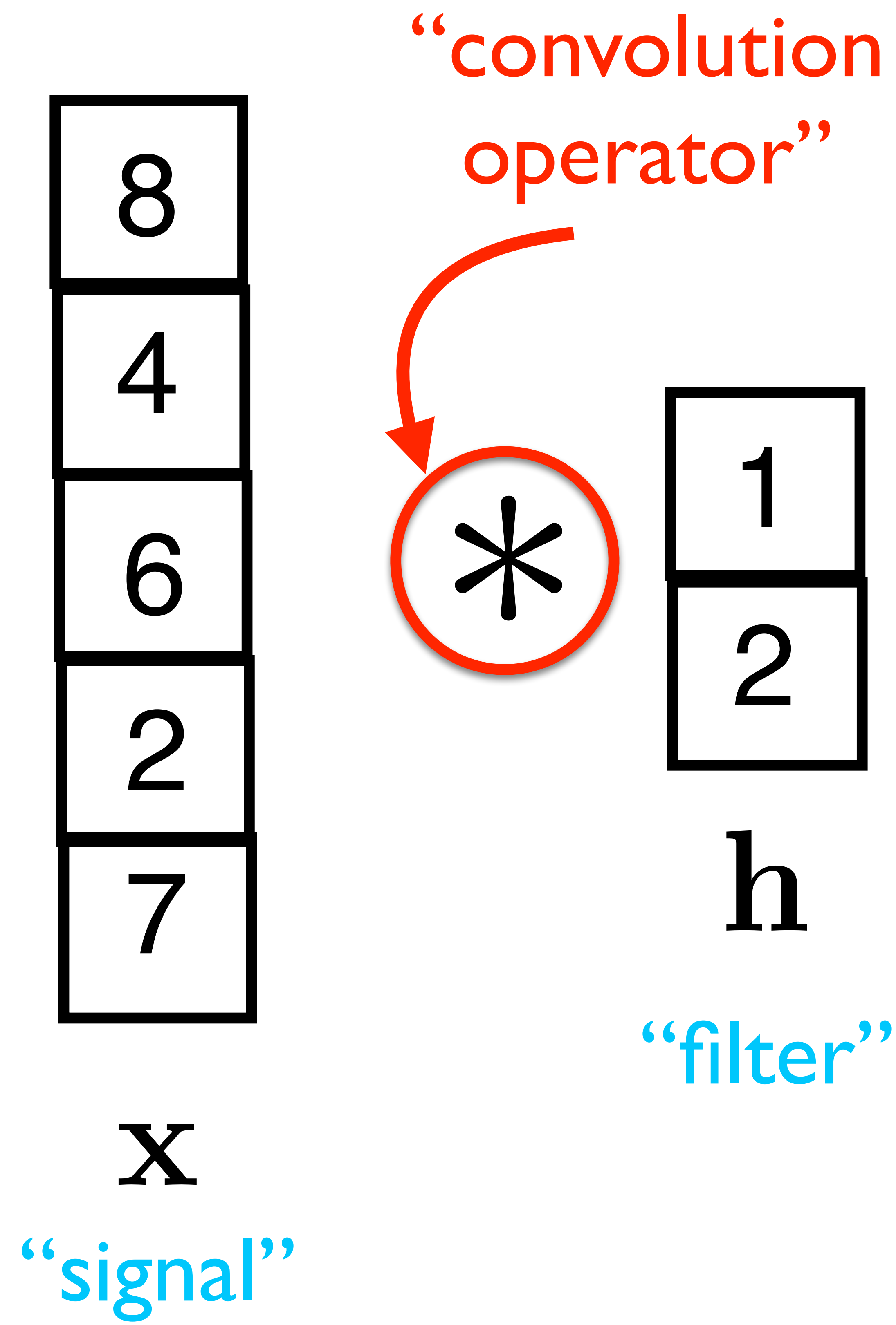
LeCun 1980

Reminder: Convolution



Reminder: Convolution

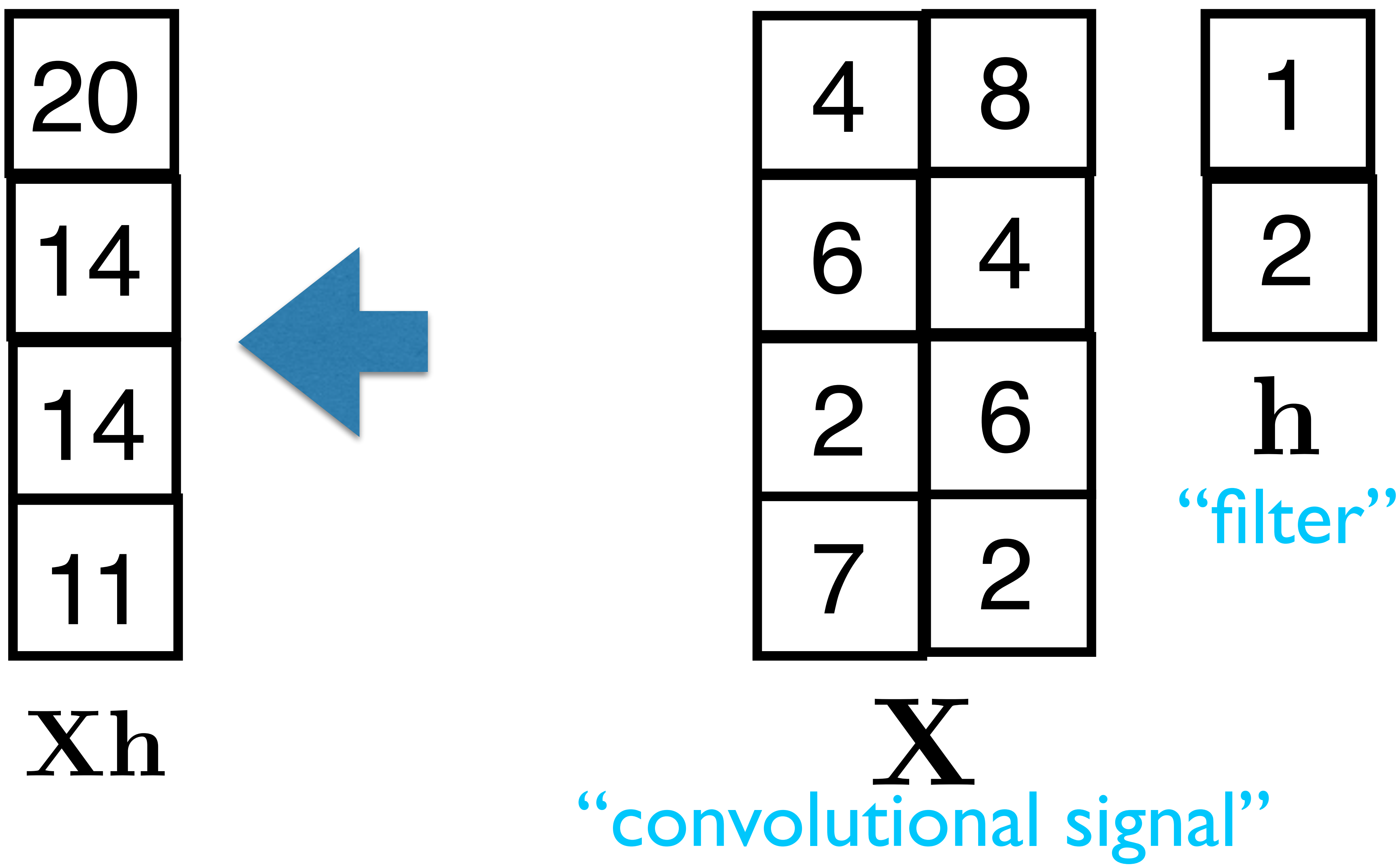
```
>> conv(x,h,'valid')
ans =

    20
    14
    14
    11
```



Reminder: Convolution

$$
\underset{\mathbf{Hx}}{\begin{bmatrix} 20 \\ 14 \\ 14 \\ 11 \end{bmatrix}}
\;\leftarrow\;
\underset{\substack{\mathbf{H} \\ \text{“convolutional matrix”}}}{\begin{bmatrix} 2 & 1 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix}}
\;
\underset{\substack{\mathbf{x} \\ \text{“signal”}}}{\begin{bmatrix} 8 \\ 4 \\ 6 \\ 2 \\ 7 \end{bmatrix}}
$$

Reminder: Convolution



Multiple Filters

$$\begin{array}{c}(D \cdot M \times 1) \\ \begin{bmatrix} \mathbf{x} * \mathbf{h}_1 \\ \vdots \\ \mathbf{x} * \mathbf{h}_M \end{bmatrix}\end{array}$$

Multiple Filters

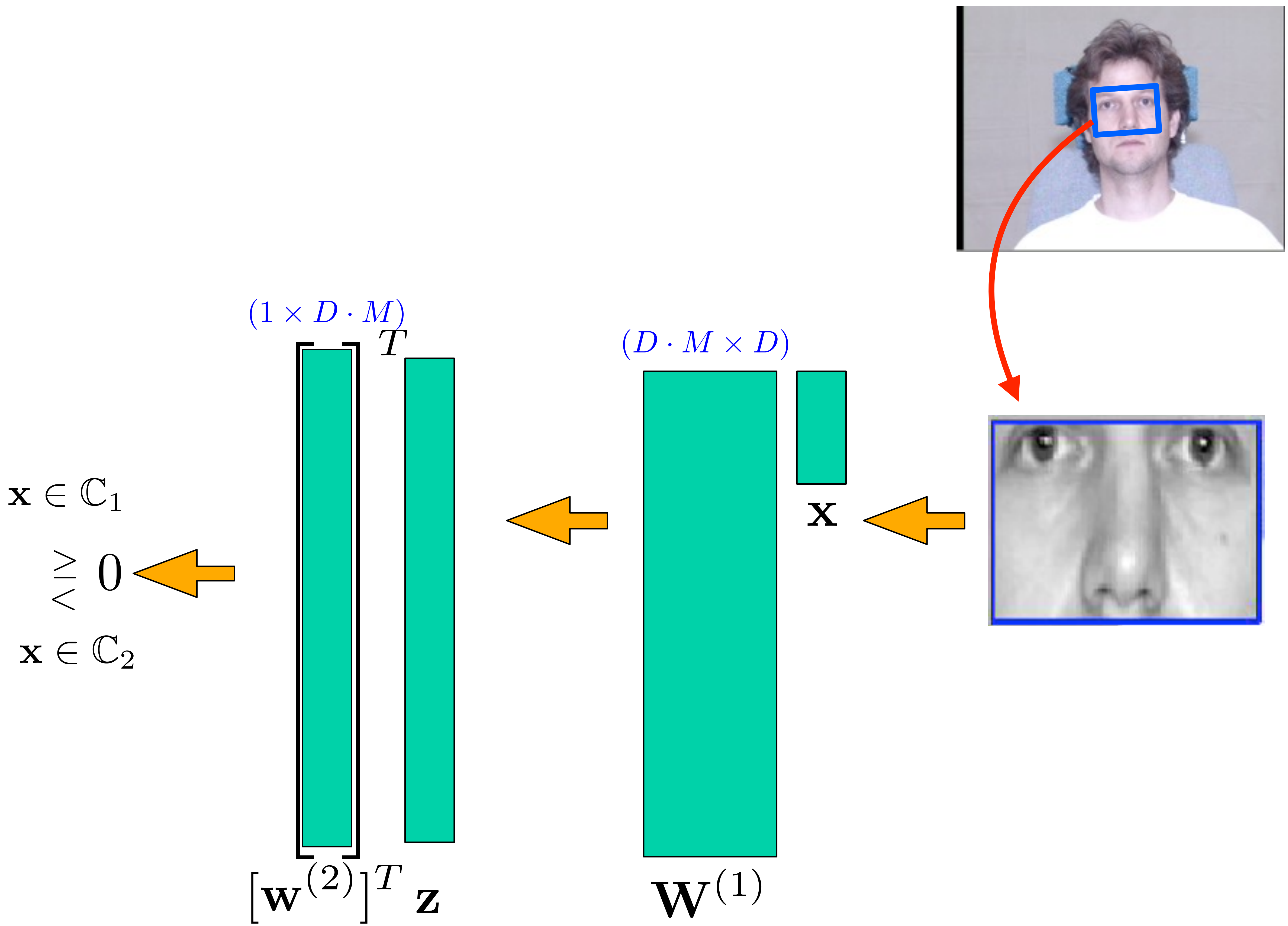
$$\overset{(D \cdot M \times D)}{\begin{bmatrix} \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_M \end{bmatrix}} \overset{(D \times 1)}{\mathbf{x}} \Leftarrow \overset{(D \cdot M \times 1)}{\begin{bmatrix} \mathbf{x} * \mathbf{h}_1 \\ \vdots \\ \mathbf{x} * \mathbf{h}_M \end{bmatrix}}$$

Multiple Filters

$$\underset{(D\cdot M\times D)}{\begin{bmatrix}\mathbf{H}_1\\ \vdots \\ \mathbf{H}_M\end{bmatrix}}\underset{(D\times 1)}{\mathbf{x}} \Leftarrow \underset{(D\cdot M\times 1)}{\begin{bmatrix}\mathbf{x}*\mathbf{h}_1\\ \vdots \\ \mathbf{x}*\mathbf{h}_M\end{bmatrix}}$$

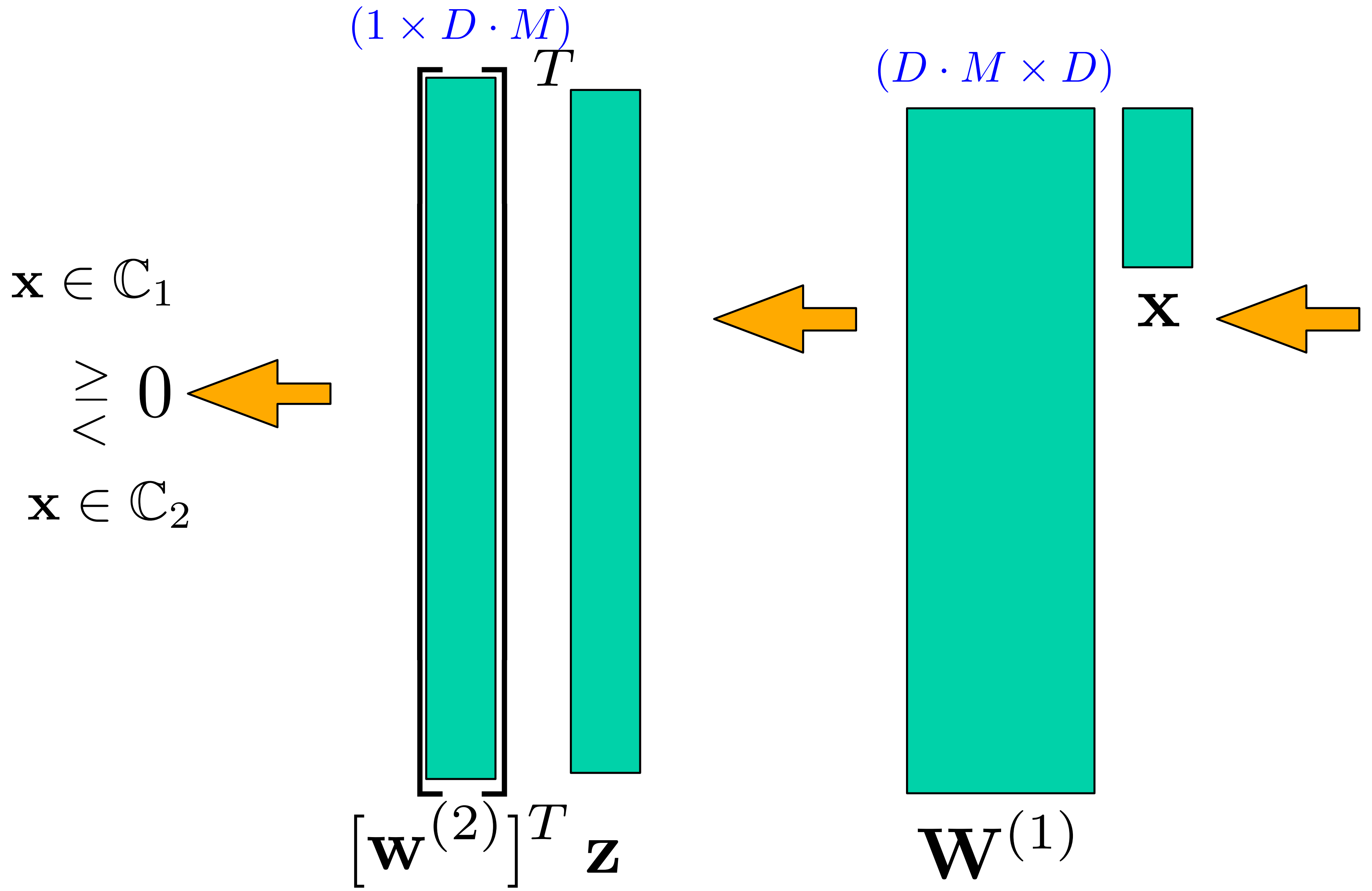
"convolution matrix"

Convolutional Neural Network



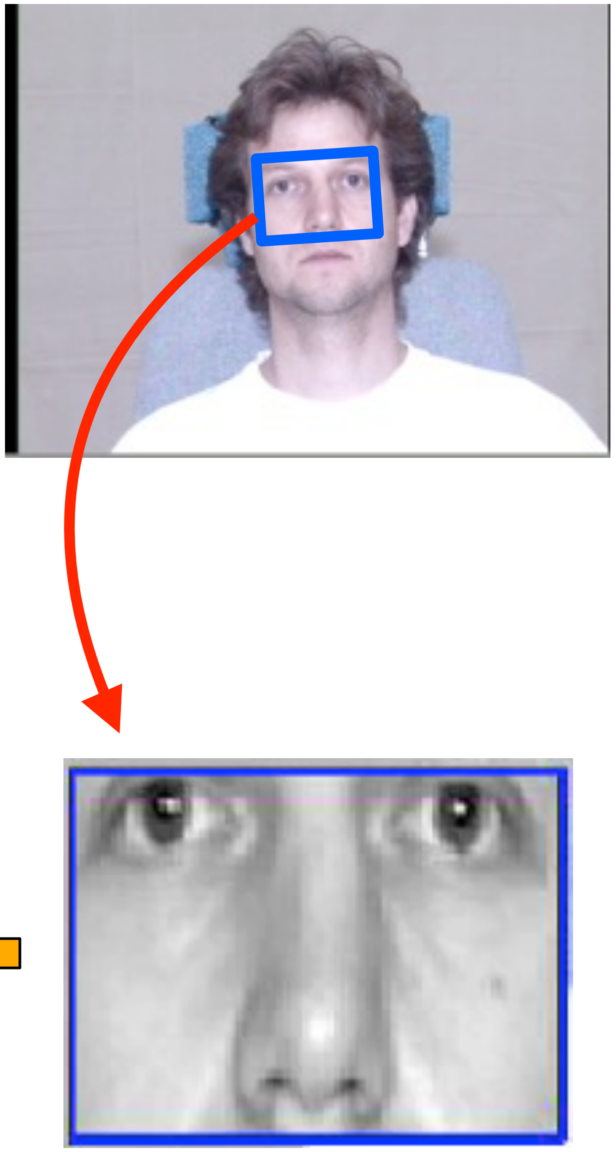
Convolutional Neural Network

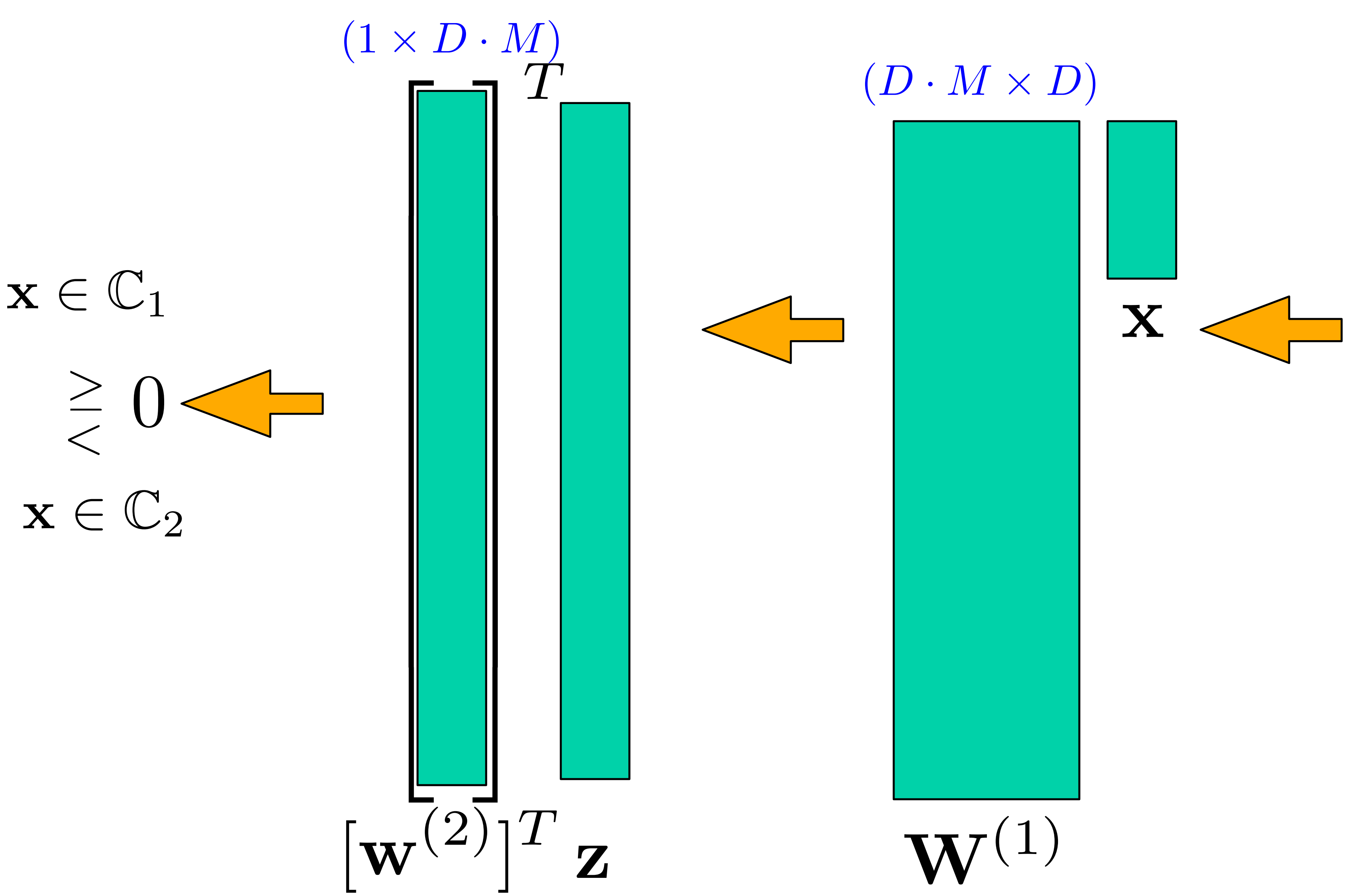
$$\mathbf{W}^{(1)}\mathbf{x} = \begin{bmatrix} \mathbf{W}_1^{(1)} \\ \vdots \\ \mathbf{W}_M^{(1)} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{x} * \mathbf{w}_1^{(1)} \\ \vdots \\ \mathbf{x} * \mathbf{w}_M^{(1)} \end{bmatrix}$$



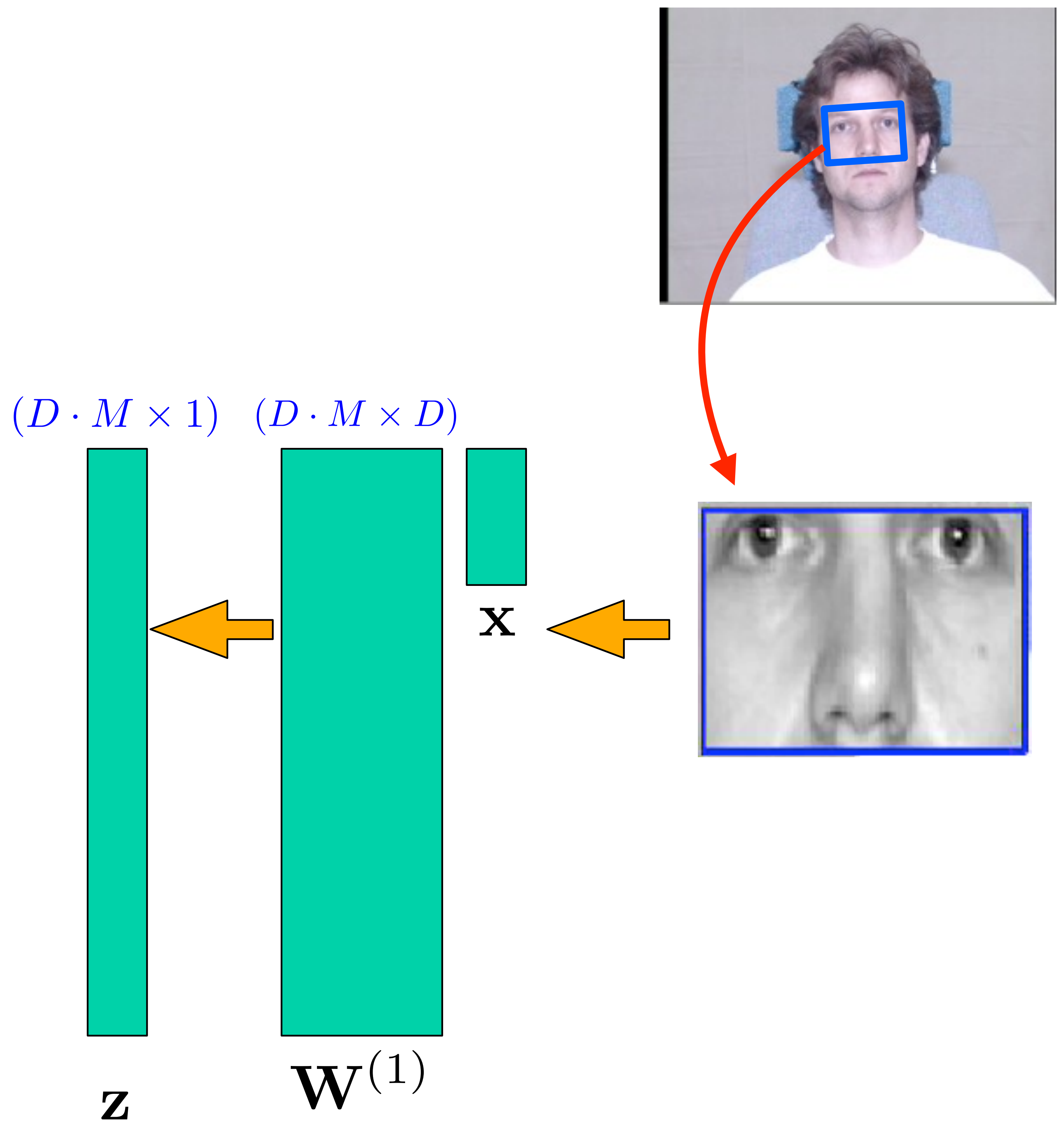
Convolutional Neural Network

$$\mathbf{z} = h[\mathbf{W}^{(1)}\mathbf{x}]$$

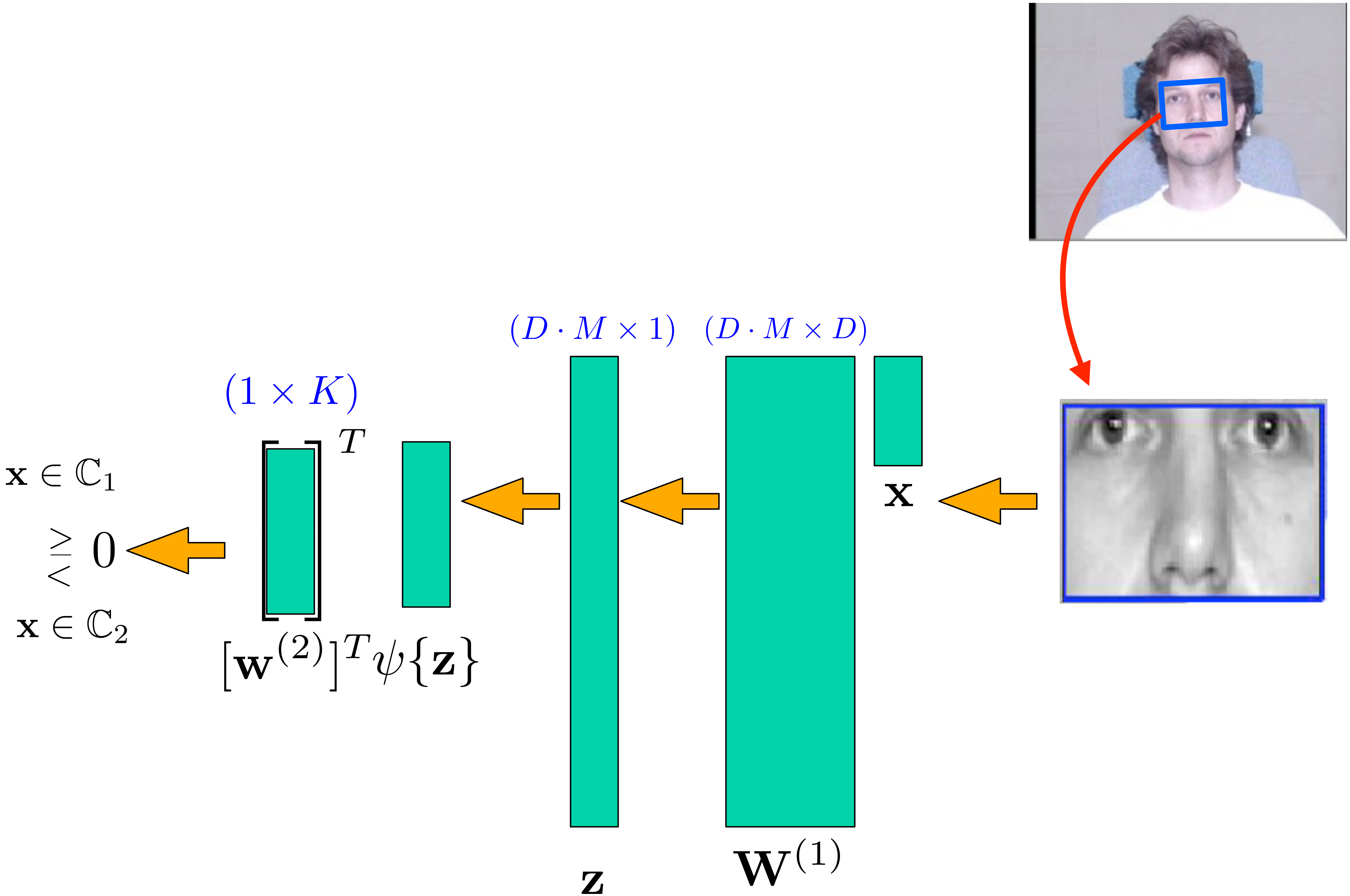




Convolutional Neural Network

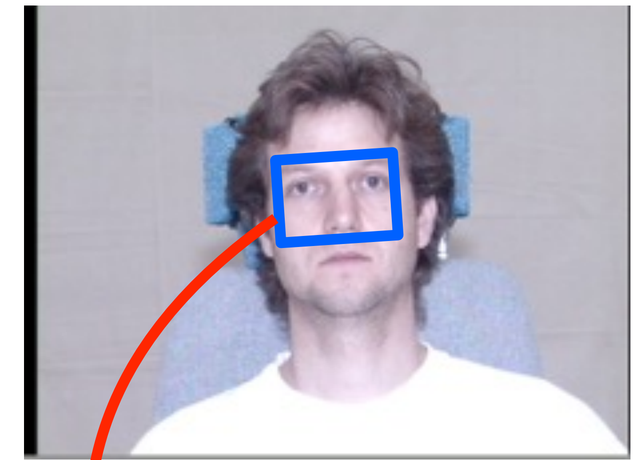


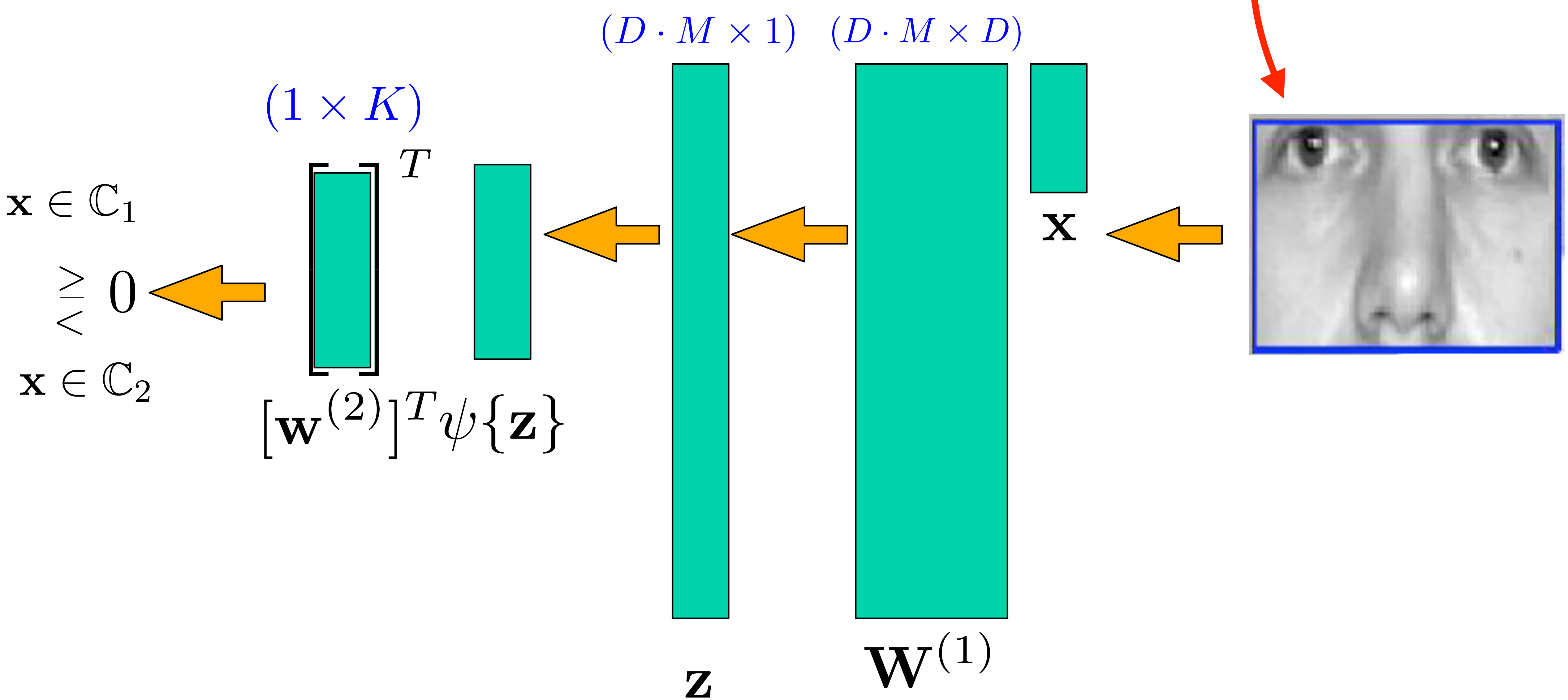
Convolutional Neural Network



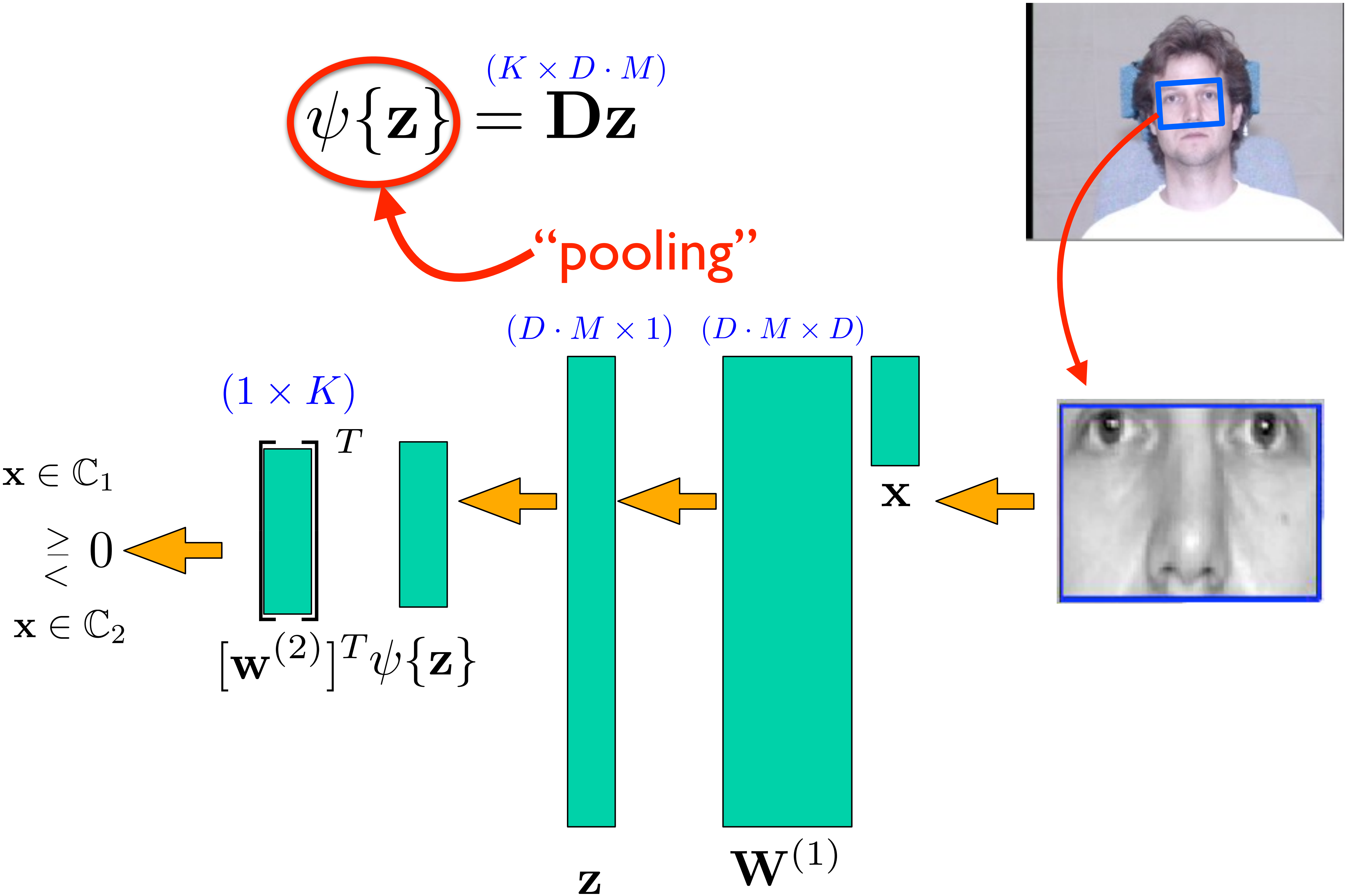
Convolutional Neural Network

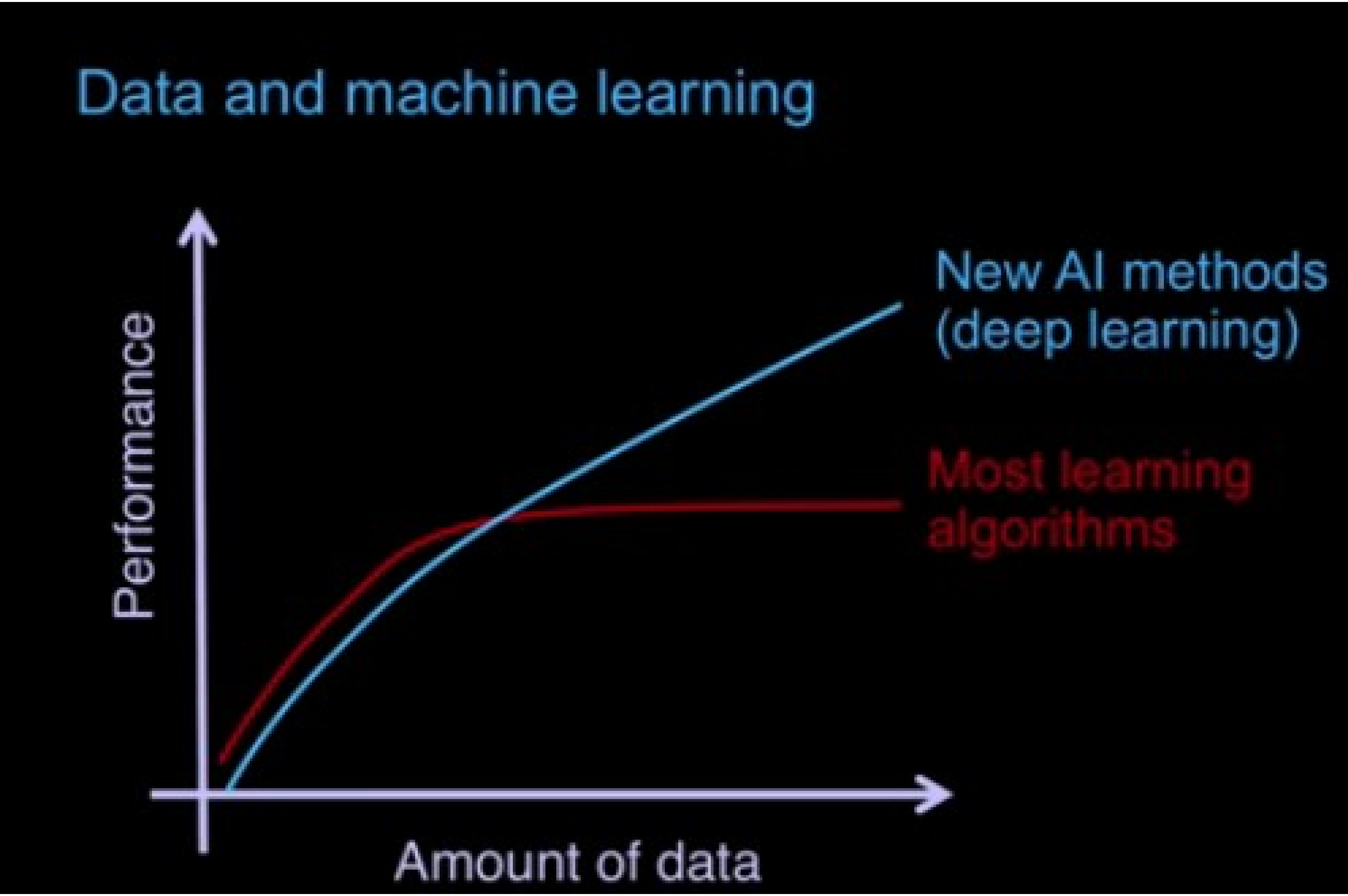
$$\psi\{\mathbf{z}\} = \overset{(K \times D \cdot M)}{\mathbf{D}}\mathbf{z}$$





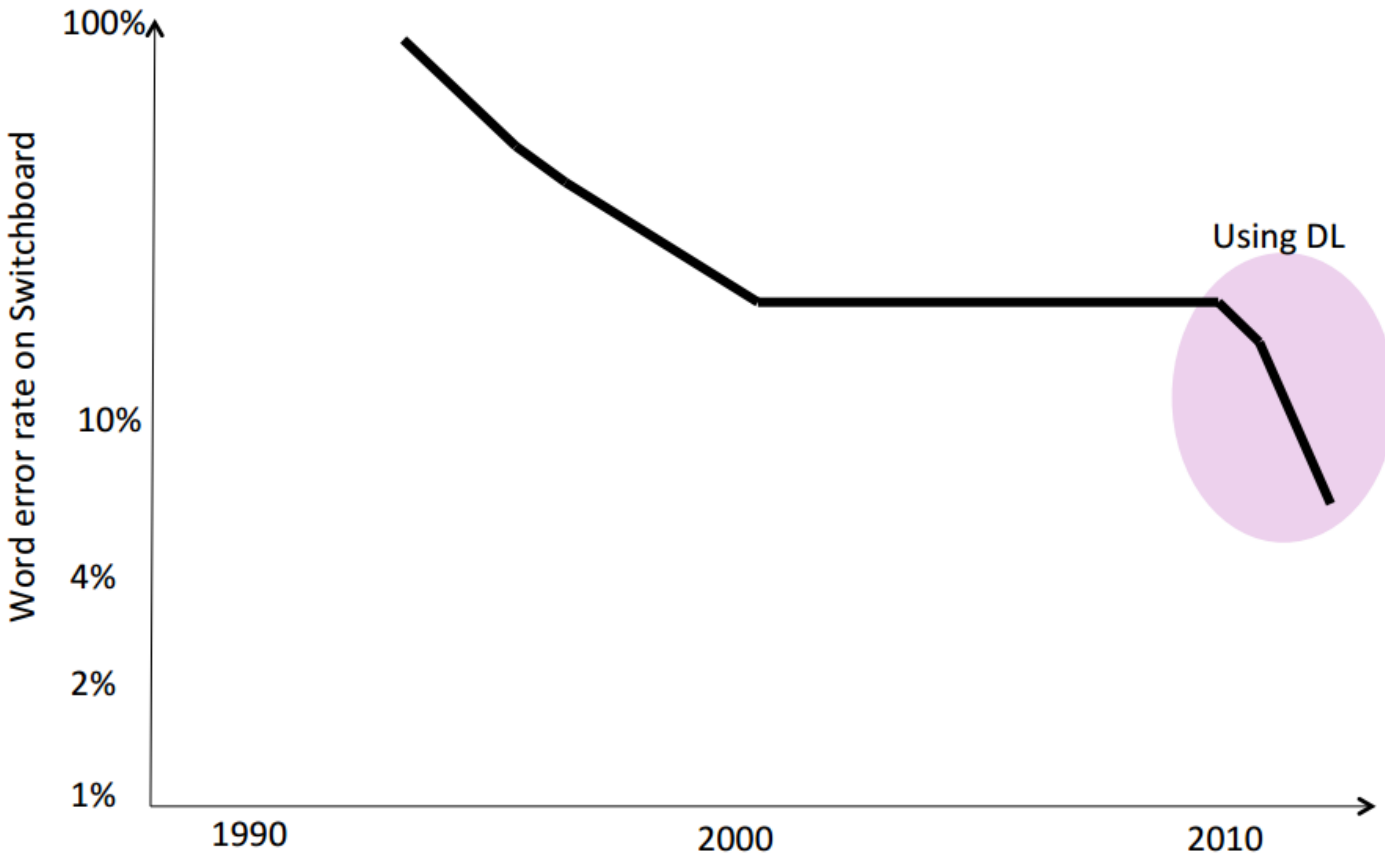
Convolutional Neural Network



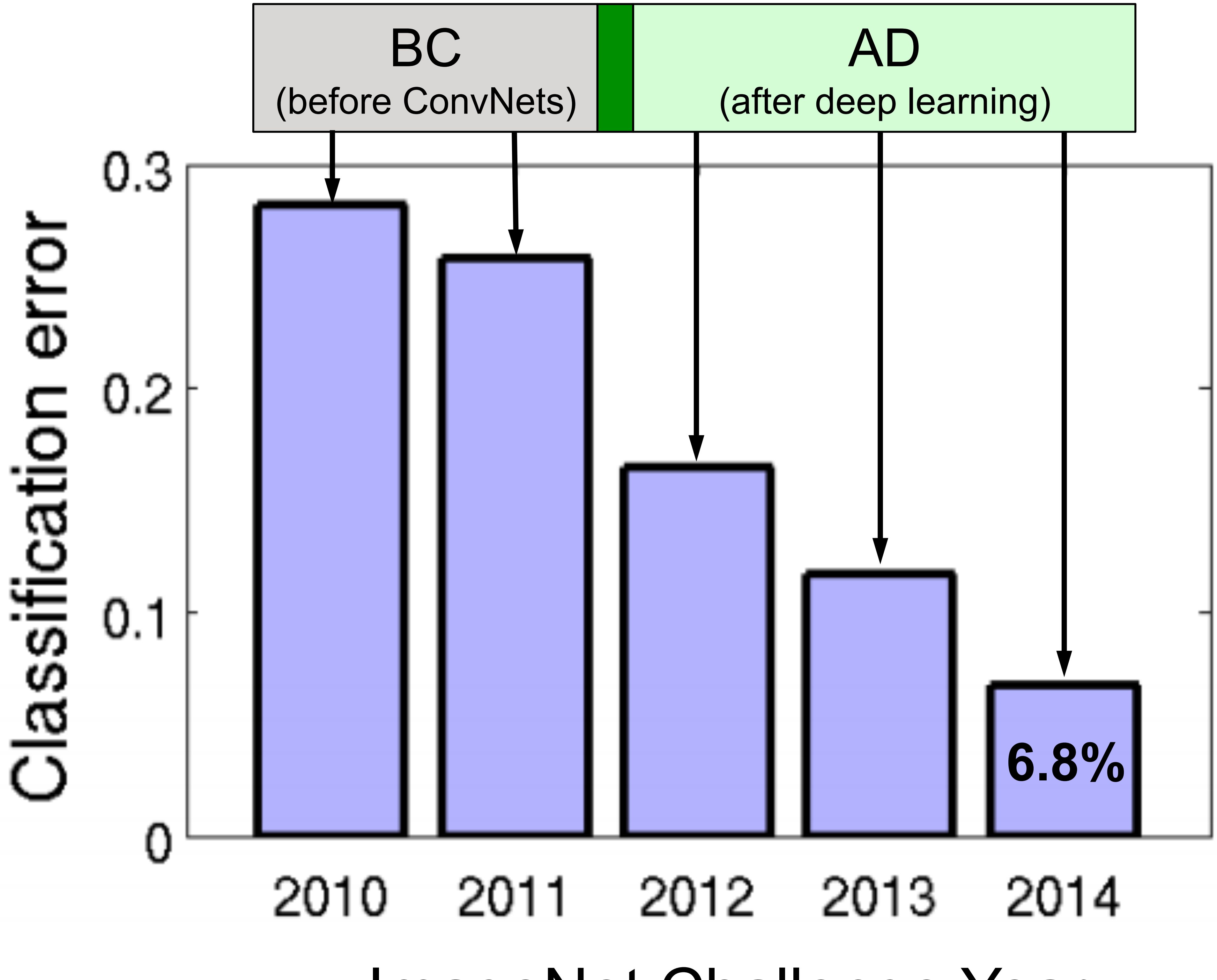


Data and machine learning
New AI methods (deep learning)
Most learning algorithms
Performance
Amount of data

Impact on Speech Recognition



Impact on Object Recognition



Audio

TIMIT Phone classification	Accuracy
Prior art (Clarkson et al.,1999)	79.6%
Feature learning	**80.3%**

TIMIT Speaker identification	Accuracy
Prior art (Reynolds, 1995)	99.7%
Feature learning	**100.0%**

Images

CIFAR Object classification	Accuracy
Prior art (Ciresan et al., 2011)	80.5%
Feature learning	**82.0%**

NORB Object classification	Accuracy
Prior art (Scherer et al., 2010)	94.4%
Feature learning	**95.0%**

Video

Hollywood2 Classification	Accuracy
Prior art (Laptev et al., 2004)	48%
Feature learning	**53%**

YouTube	Accuracy
Prior art (Liu et al., 2009)	71.2%
Feature learning	**75.8%**

KTH	Accuracy
Prior art (Wang et al., 2010)	92.1%
Feature learning	**93.9%**

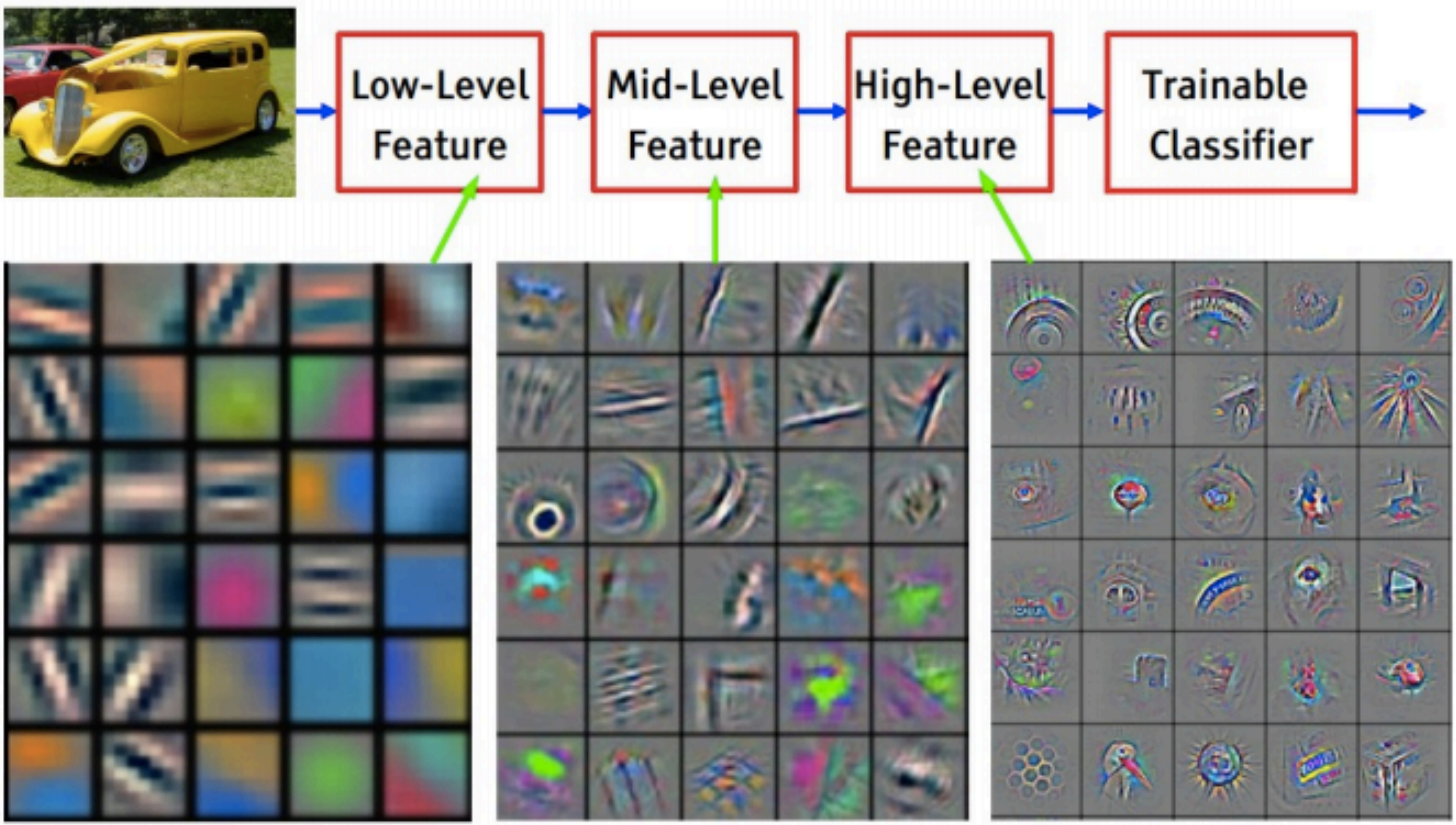
UCF	Accuracy
Prior art (Wang et al., 2010)	85.6%
Feature learning	**86.5%**

Text/NLP

Paraphrase detection	Accuracy
Prior art (Das & Smith, 2009)	76.1%
Feature learning	**76.4%**

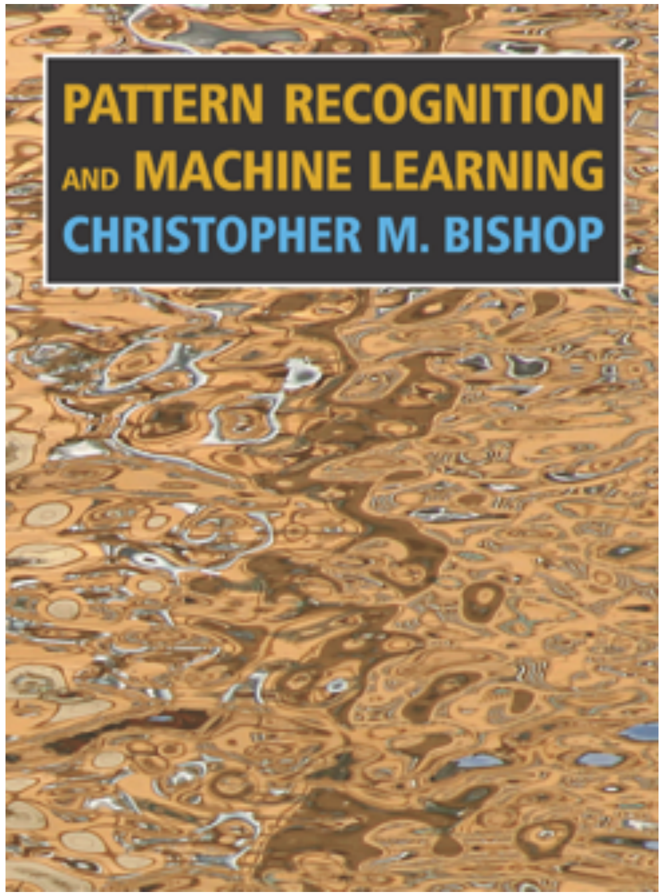
Sentiment (MR/MPQA data)	Accuracy
Prior art (Nakagawa et al., 2010)	77.3%
Feature learning	**77.7%**

Visualizing CNNs



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

More to read…



- Bishop "Pattern Recognition and Machine Learning", 2006. Chapter 5.