Classificação de Padrões com Redes Convolucionais

Jefersson Alex dos Santos

jefersson@dcc.ufmg.br

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

Roteiro da Aula

UFMG

Pattern Classifier

UFMG

Typical Steps

Building

Pattern Classifier

UFMG

Typical Steps

Building

Example

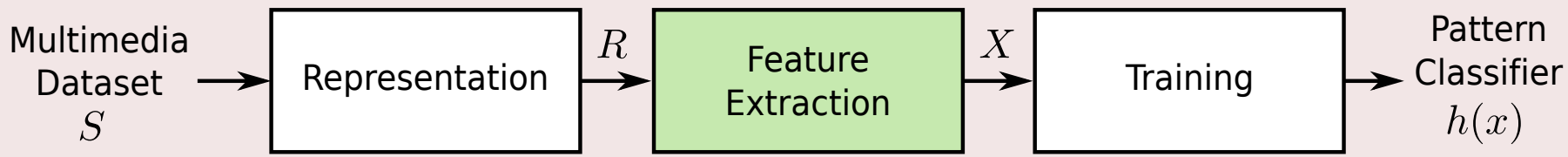
- pixel
- window
- block
- patch
- segmented region
- ...

Pattern Classifier

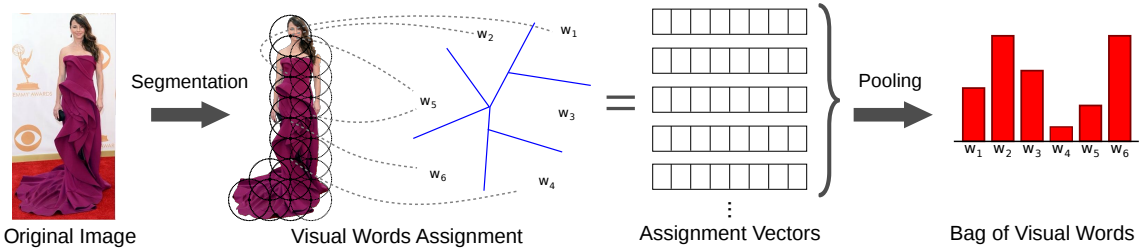
UFMG

Typical Steps

Building



Example

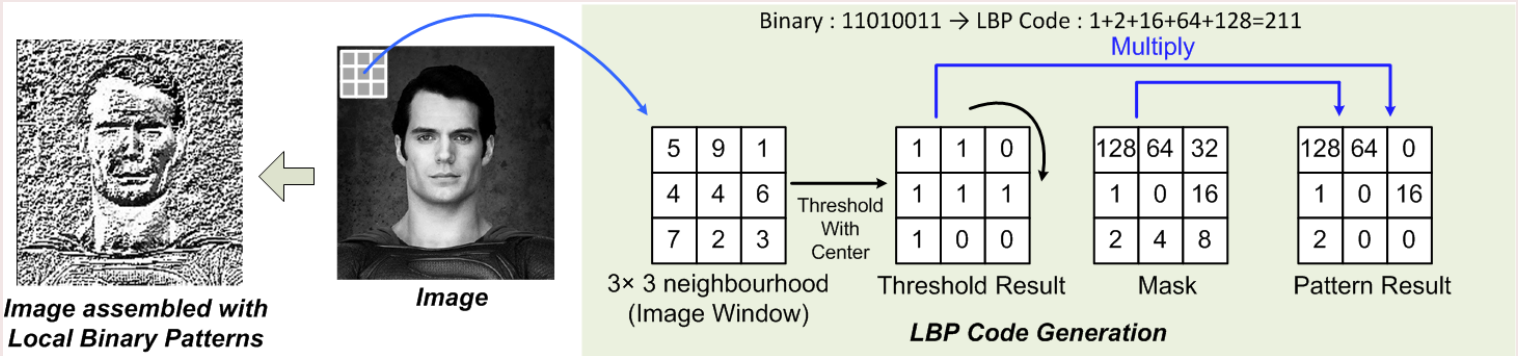


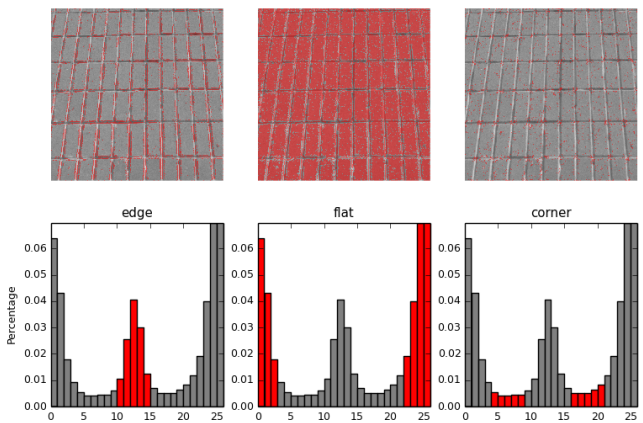
- Color Histogram
- BoW + SIFT
- ...

Pattern Classifier

Feature Extracion Strategies

Example: Local Binary Patterns (LBP)





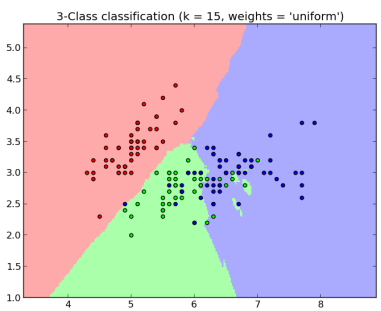
Pattern Classifier

UFMG

Typical Steps

Building

Example



- Artificial Neural Networks
- Support Vector Machines
- Nearest Neighbors
- ...

Pattern Classifier

UFMG

Typical Steps

Building

How to use the classifier?

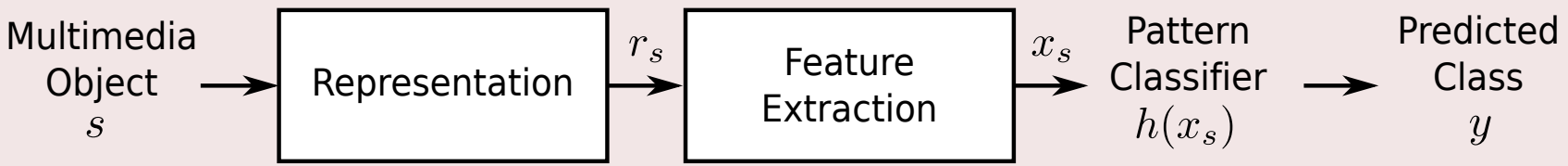
Pattern Classifier

UFMG

Typical Steps

Building

Using

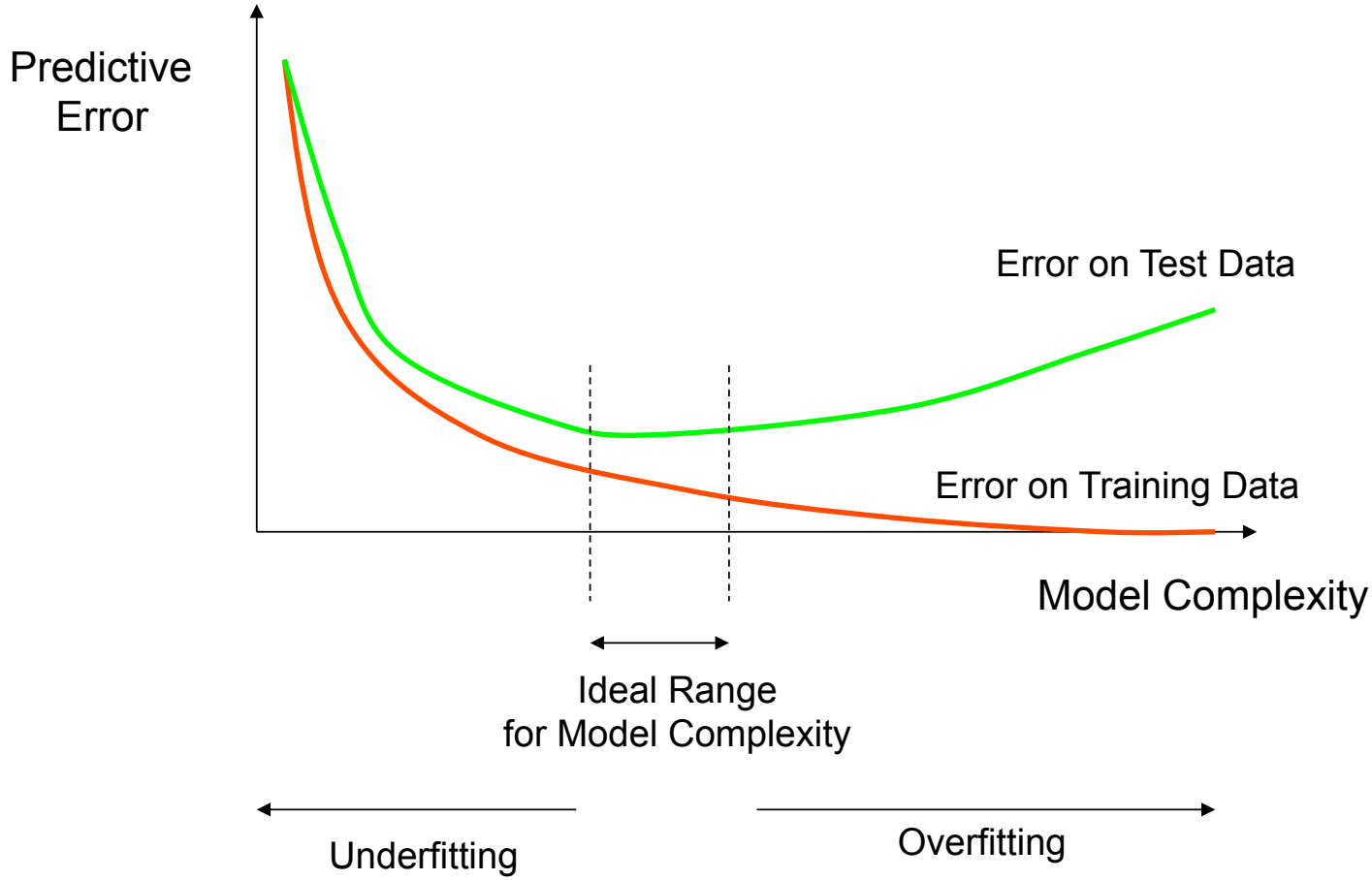


Pattern Classifier

Some Quality Factors

- Reasonable data representation
- Suitable feature extraction
- Effective learning algorithms
- Representative training data

How Overfitting affects Prediction



Roteiro da Aula

UFMG

UFMG

Slides Notes:

- Some slides are based on materials collected from other researchers.
- This class specifically uses slides prepared by **Keiller Nogueira**, by **Prof. Yoshua Bengio** (U. Montreal), by **Prof. Geoffrey Hinton** (U. Toronto), and by **Prof. Yann Lecun** (New York University).

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks
- Why not propose algorithms that simulates this learning?

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks
- Why not propose algorithms that simulates this learning?
  - Algorithms

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks
- Why not propose algorithms that simulates this learning?
  - Algorithms $\rightarrow$ brain

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks
- Why not propose algorithms that simulates this learning?
  - Algorithms $\rightarrow$ brain $\rightarrow$ neurons

Artificial Neural Networks

UFMG

- Brain: learns very well when doing some tasks
- Why not propose algorithms that simulates this learning?
  - Algorithms $\rightarrow$ brain $\rightarrow$ neurons

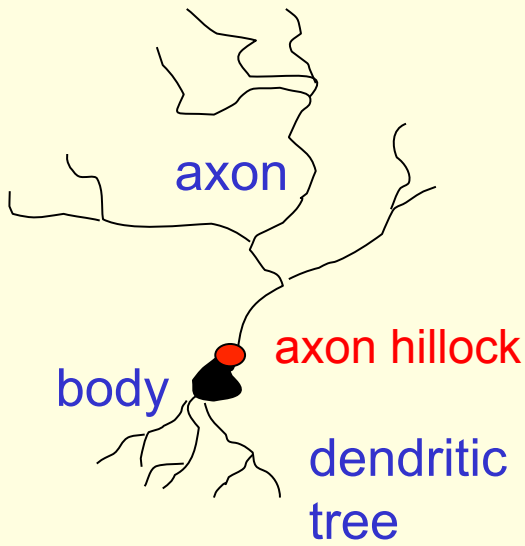
This study has started a long time ago!

Reasons to study neural computation

- To understand how the brain actually works.
  - Its very big and very complicated and made of stuff that dies when you poke it around. So we need to use computer simulations.
- To understand a style of parallel computation inspired by neurons and their adaptive connections.
  - Very different style from sequential computation.
    - should be good for things that brains are good at (e.g. vision)
    - Should be bad for things that brains are bad at (e.g. 23 x 71)
- To solve practical problems by using novel learning algorithms inspired by the brain (this course)
  - Learning algorithms can be very useful even if they are not how the brain actually works.

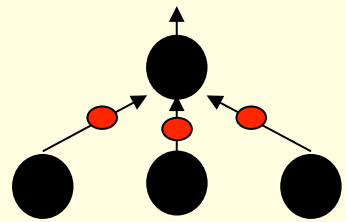
A typical cortical neuron

- Gross physical structure:
  - There is one axon that branches
  - There is a dendritic tree that collects input from other neurons.
- Axons typically contact dendritic trees at synapses
  - A spike of activity in the axon causes charge to be injected into the post-synaptic neuron.
- Spike generation:
  - There is an axon hillock that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarize the cell membrane.



How the brain works on one slide!

- Each neuron receives inputs from other neurons
  - A few neurons also connect to receptors.
  - Cortical neurons use spikes to communicate.
- The effect of each input line on the neuron is controlled by a synaptic weight
  - The weights can be positive or negative.
- The synaptic weights adapt so that the whole network learns to perform useful computations
  - Recognizing objects, understanding language, making plans, controlling the body.
- You have about 10^{11} neurons each with about 10^{4} weights.
  - A huge number of weights can affect the computation in a very short time. Much better bandwidth than a workstation.



Artificial Neural Networks

UFMG

Systems of interconnected “neurons” which can compute values from inputs

- Several types of “neurons” were proposed along with different NN and architectures
- Some of them are used until today

UFMG

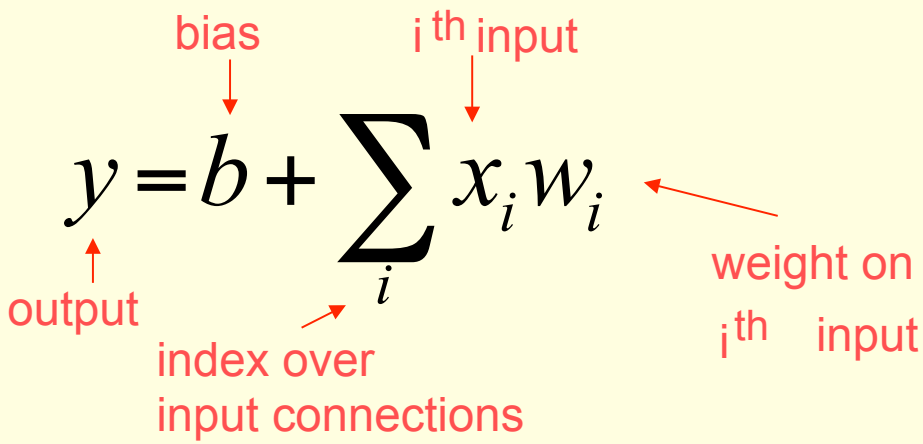
Neural Networks

Some History

- 1943: first computational model for neural networks using mathematics and algorithms
- 1958: Perceptrons
- 1975: Backpropagation algorithm
- 1980s: beginning of Recurrent Neural Networks (RNN)
- 1980: first Convolutional Neural Networks (CNN)
- 1982: Hopfiled (RNN)
- 1986: Harmonium (initial model of Restricted Boltzmann Machine (RBM))
- 1987: rediscovered Backpropagation algorithm
- 1998: improved Convolutional Neural Networks (more similar to current model, LeCun)
- 2005: faster learning algorithms for what we know as RBM
- 2008: auto-encoders (AE)
- 2011: denoising auto-encoders (D-AE)

Linear neurons

- These are simple but computationally limited
  - If we can make them learn we may get insight into more complicated neurons.



Binary threshold neurons

- There are two equivalent ways to write the equations for a binary threshold neuron:

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq \theta \\ 0 \text{ otherwise} \end{cases}$$

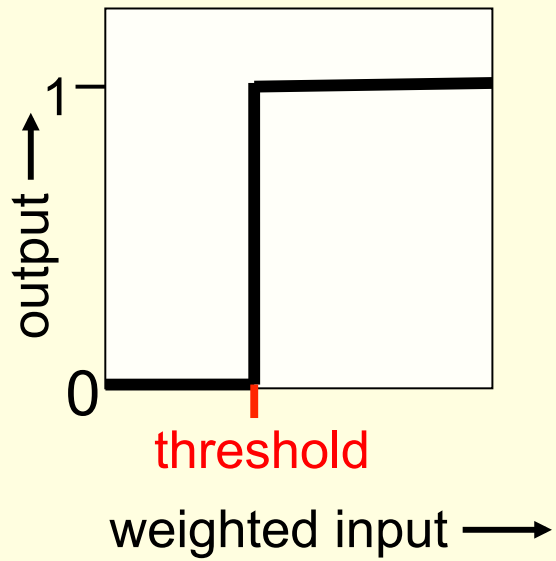
$$\theta = -b$$

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

Binary threshold neurons

- McCulloch-Pitts (1943): influenced Von Neumann.
  - First compute a weighted sum of the inputs.
  - Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
  - McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition!



Activation Functions

UFMG

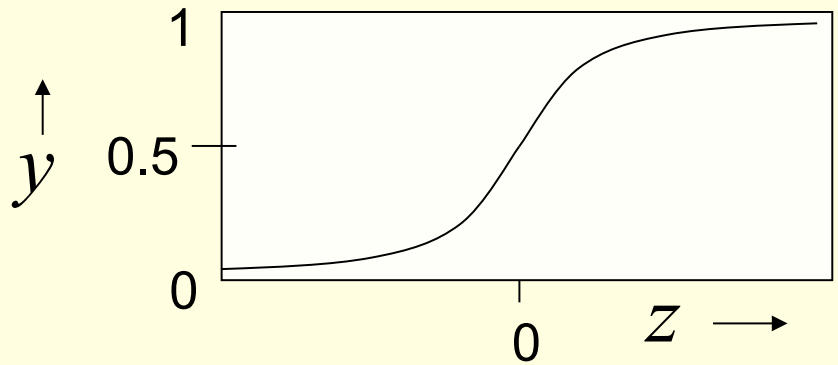
Some examples:

- Sigmoid
- ReLU

Sigmoid neurons

- These give a real-valued output that is a smooth and bounded function of their total input.
  - Typically they use the logistic function
  - They have nice derivatives which make learning easy (see lecture 3).

$$z = b + \sum_i x_i w_i \qquad y = \frac{1}{1 + e^{-z}}$$

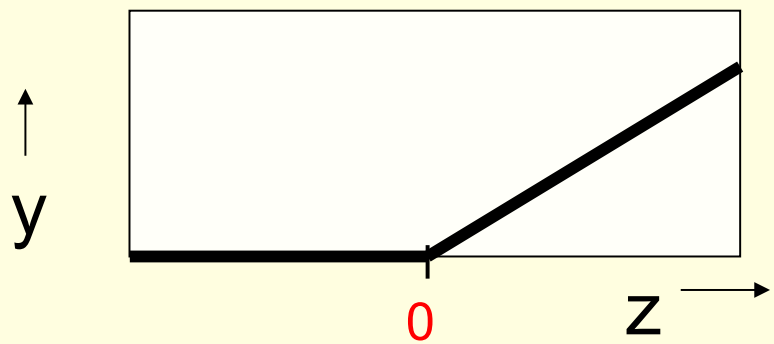


Rectified Linear Neurons
(sometimes called linear threshold neurons)

They compute a linear weighted sum of their inputs.
The output is a non-linear function of the total input.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

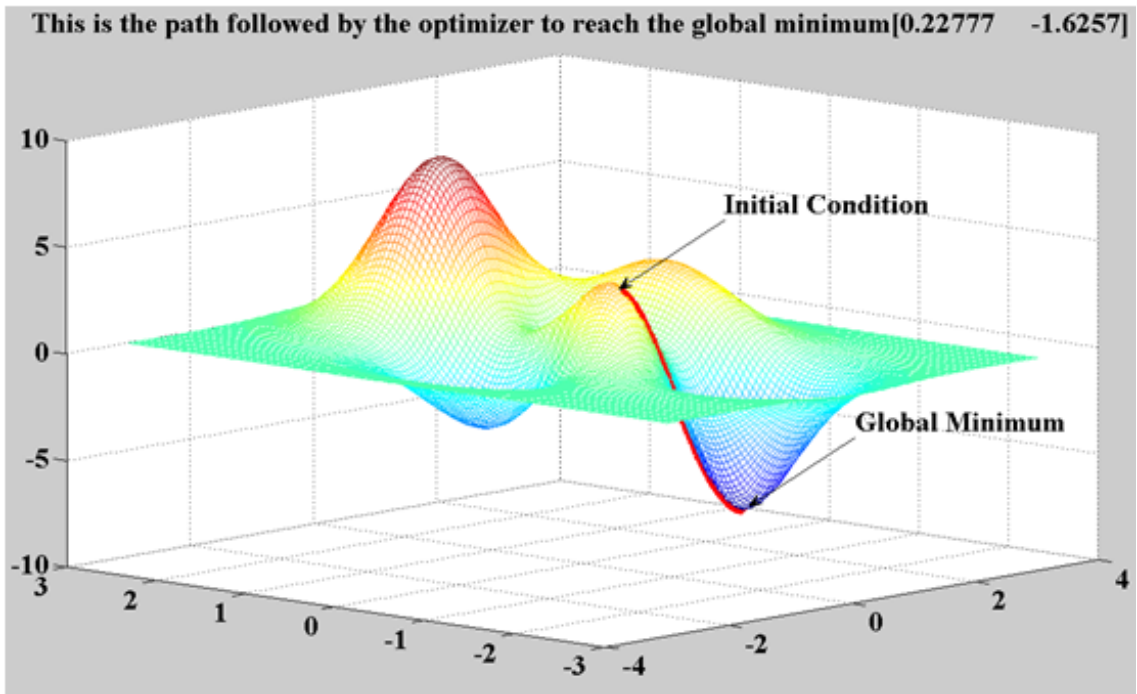


Training a Perceptron

- Optimization problem

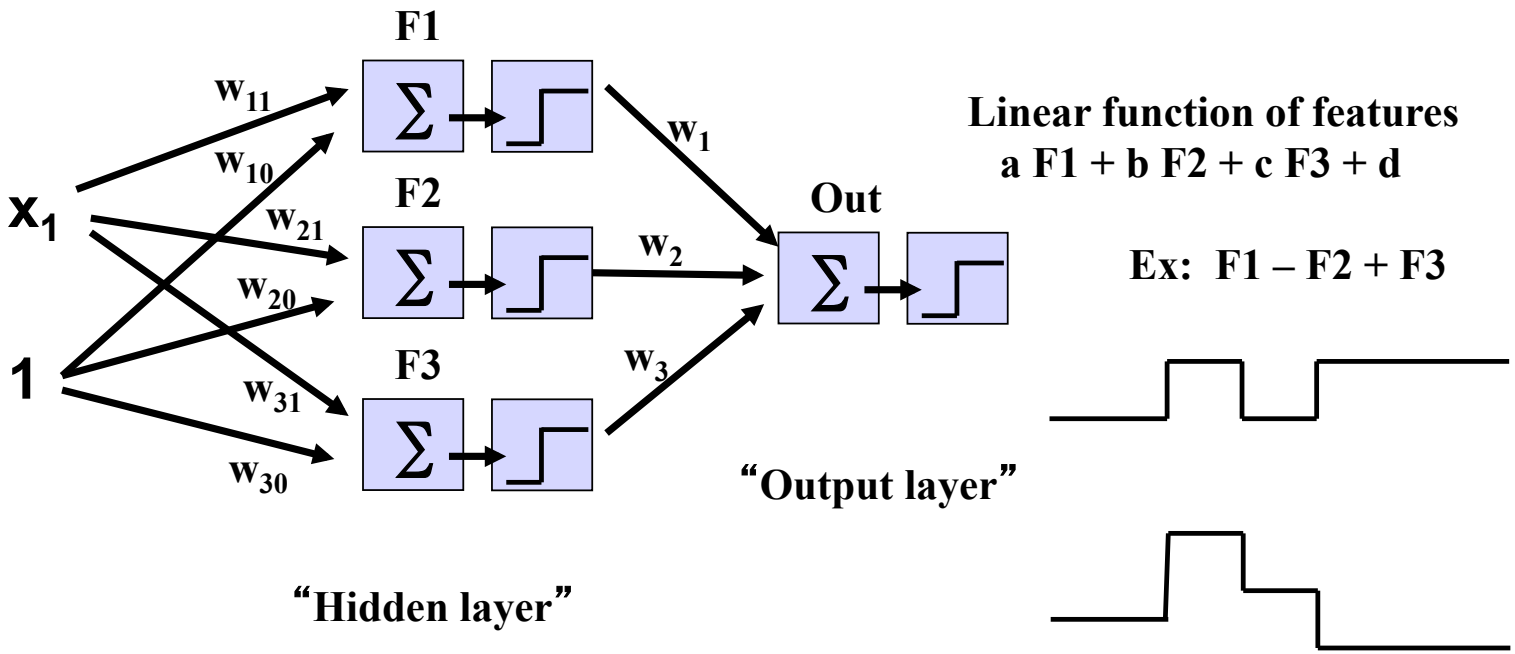
$$C(w,b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2. \tag{6}$$

- Many algorithms can be used
- Most common: Gradient descent algorithm



Multi-layer perceptron model

- Step functions are just perceptrons!
  - "Features" are outputs of a perceptron
  - Combination of features output of another



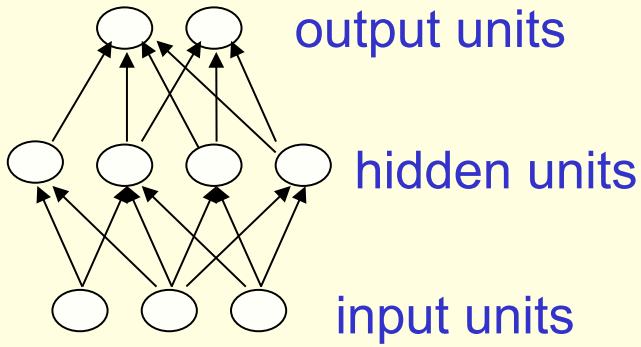
Architectures

UFMG

- Feed-forward Neural Networks
- Recurrent Networks

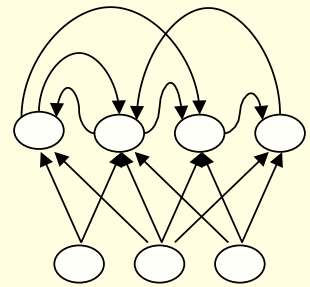
Feed-forward neural networks

- These are the commonest type of neural network in practical applications.
  - The first layer is the input and the last layer is the output.
  - If there is more than one hidden layer, we call them “deep” neural networks.
- They compute a series of transformations that change the similarities between cases.
  - The activities of the neurons in each layer are a non-linear function of the activities in the layer below.



Recurrent networks

- These have directed cycles in their connection graph.
  - That means you can sometimes get back to where you started by following the arrows.
- They can have complicated dynamics and this can make them very difficult to train.
  - There is a lot of interest at present in finding efficient ways of training recurrent nets.
- They are more biologically realistic.

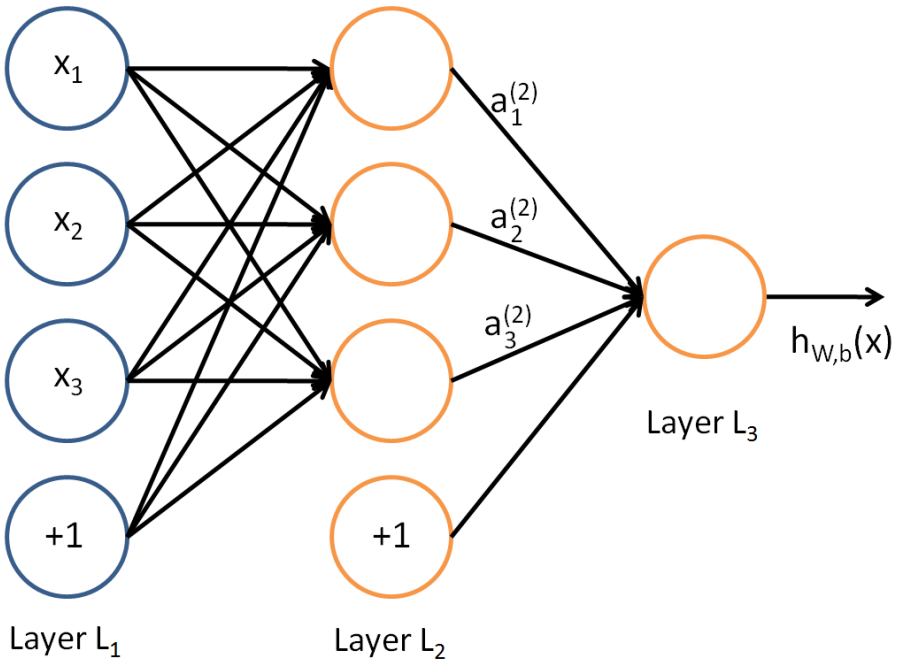


Recurrent nets with multiple hidden layers are just a special case that has some of the hidden→hidden connections missing.

Multilayer Neural Networks

UFMG

- The output of a layer is the input of other
- A cost function is used to evaluate the NN

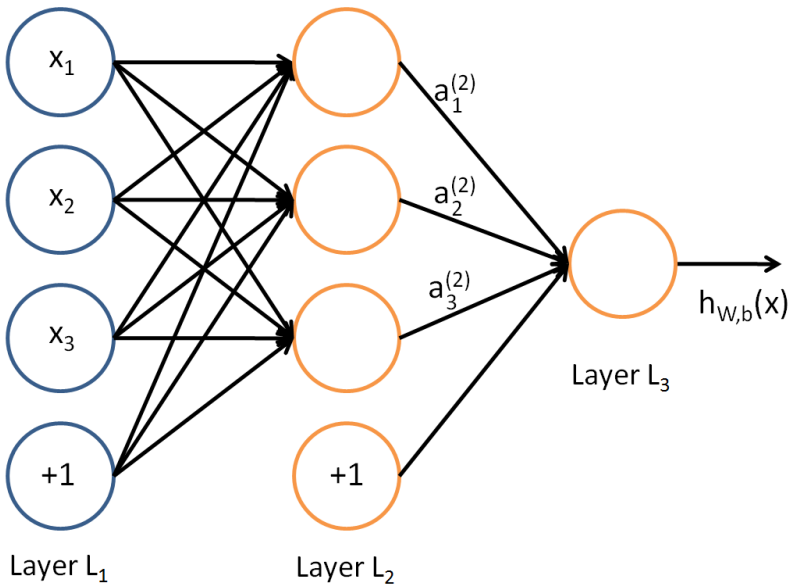


Multilayer Neural Networks

Forward Step

- Sequential computation

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$
$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$h_{W,b}(x) = a^{(3)} = f(z^{(3)})$$



- Recursive computation

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$
$$a^{(l+1)} = f(z^{(l+1)})$$

Multilayer Neural Networks

Forward Step

Some issues:

- Is the forward step enough for training a classifier?

Multilayer Neural Networks

Forward Step

Some issues:

- Is the forward step enough for training a classifier?
- What about the error generated by the output?

Multilayer Neural Networks

Forward Step

Some issues:

- Is the forward step enough for training a classifier?
- What about the error generated by the output?
- What if we can update the weights to increase the accuracy?

The idea behind backpropagation

- We don't know what the hidden units ought to do, but we can compute how fast the error changes as we change a hidden activity.
  - Instead of using desired activities to train the hidden units, use error derivatives w.r.t. hidden activities.
  - Each hidden activity can affect many output units and can therefore have many separate effects on the error. These effects must be combined.
- We can compute error derivatives for all the hidden units efficiently at the same time.
  - Once we have the error derivatives for the hidden activities, its easy to get the error derivatives for the weights going into a hidden unit.

Backpropagation Algorithm

UFMG

- It updates the weight function based on errors of forward layer

Backpropagation Algorithm

UFMG

- It updates the weight function based on errors of forward layer
- It computes the partial derivative
  - Easier to obtain
  - Generally based on Gradient Descent algorithm [**?**]

Backpropagation Algorithm

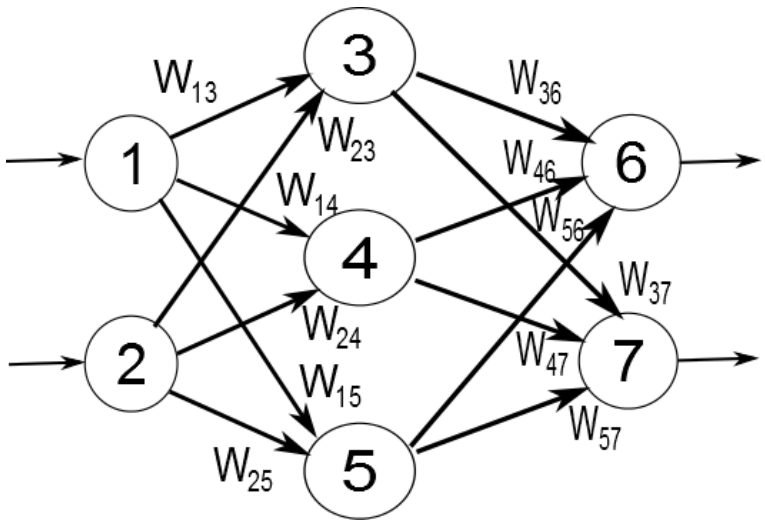
UFMG

1. Perform a forward pass
2. For each output unit, calculate the error
3. Propagate the error through hidden layers
4. Compute the partial derivatives for weights and bias
5. Use Gradient Descent algorithm to minimize the error

Backpropagation Algorithm

UFMG

Example

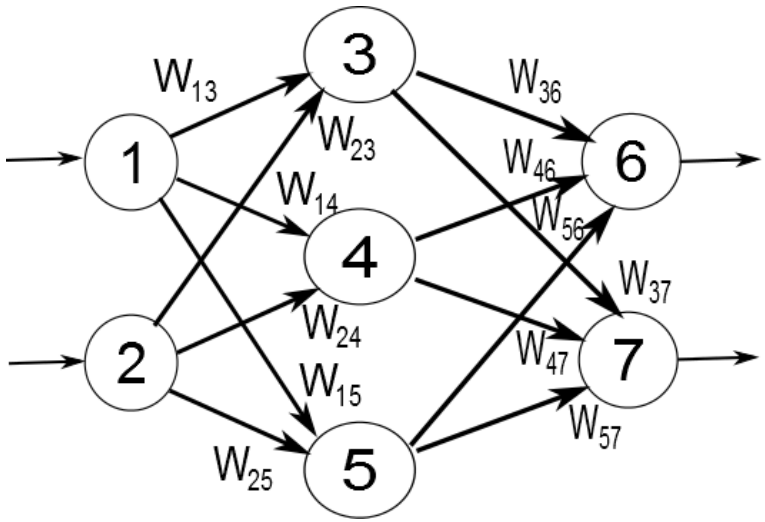


- Cost Function (based on squared error): $J(W,x,y) = \frac{1}{2}(target - a(W,x))^2$
- Use sigmoid

UFMG

Backpropagation Algorithm

Example



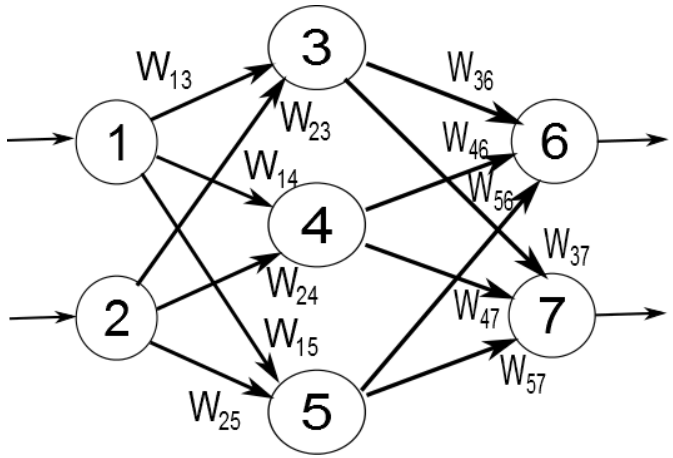
- Feedforward pass:

$$a^{(3)} = f(z^{(3)}) = f((W_{13} * a^{(1)}) + (W_{23} * a^{(2)}))$$
$$a^{(4)} = f(z^{(4)}) = f((W_{14} * a^{(1)}) + (W_{24} * a^{(2)}))$$
$$a^{(5)} = f(z^{(5)}) = f((W_{15} * a^{(1)}) + (W_{25} * a^{(2)}))$$
$$a^{(6)} = f(z^{(6)}) = f((W_{36} * a^{(3)}) + (W_{46} * a^{(4)}) + (W_{56} * a^{(5)}))$$
$$a^{(7)} = f(z^{(7)}) = f((W_{37} * a^{(3)}) + (W_{47} * a^{(4)}) + (W_{57} * x^{(5)}))$$

UFMG

Backpropagation Algorithm

Example

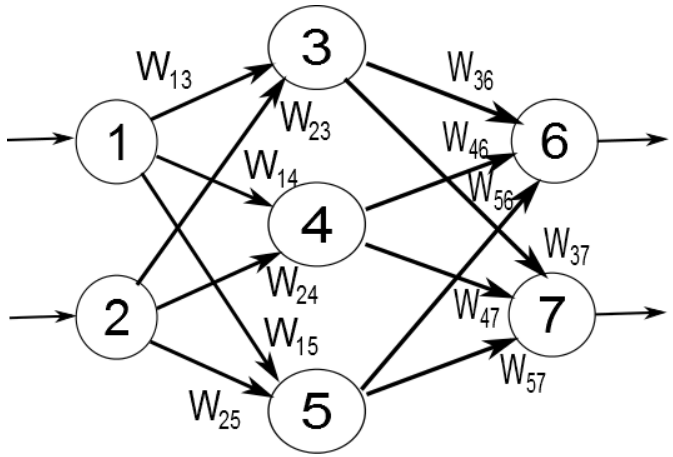


- Error for **output** layer:
  $\delta_6 = [-a^{(6)} * (1 - a^{(6)})] * (target - a^{(6)})$
  $\delta_7 = [-a^{(7)} * (1 - a^{(7)})] * (target - a^{(7)})$
- Update the **output** weights:
  $W_{36}^{+} = W_{36} - \alpha \delta_6 a^{(3)}$
  $W_{46}^{+} = W_{46} - \alpha \delta_6 a^{(4)}$
  $W_{56}^{+} = W_{56} - \alpha \delta_6 a^{(5)}$

UFMG

Backpropagation Algorithm

Example



- Error for **hidden** layers:

$$\delta_3 = [-a^{(3)} * (1 - a^{(3)})] * (\delta_6 * W_{36} + \delta_7 * W_{37})$$
$$\delta_4 = [-a^{(4)} * (1 - a^{(4)})] * (\delta_6 * W_{46} + \delta_7 * W_{47})$$
$$\delta_5 = [-a^{(5)} * (1 - a^{(5)})] * (\delta_6 * W_{56} + \delta_7 * W_{57})$$

- Update the **output** weights:

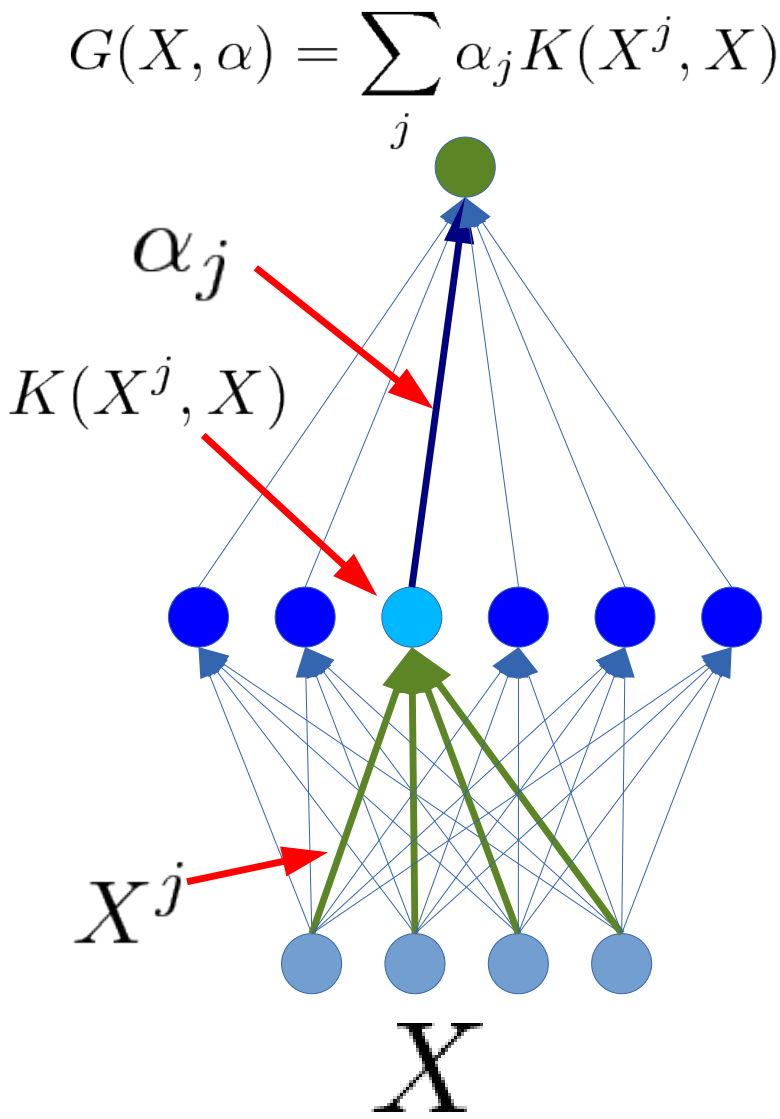
$$W_{13}^{+} = W_{13} - \alpha \delta_3 a^{(1)}$$
$$W_{14}^{+} = W_{14} - \alpha \delta_4 a^{(1)}$$
$$W_{15}^{+} = W_{15} - \alpha \delta_5 a^{(1)}$$

Roteiro da Aula

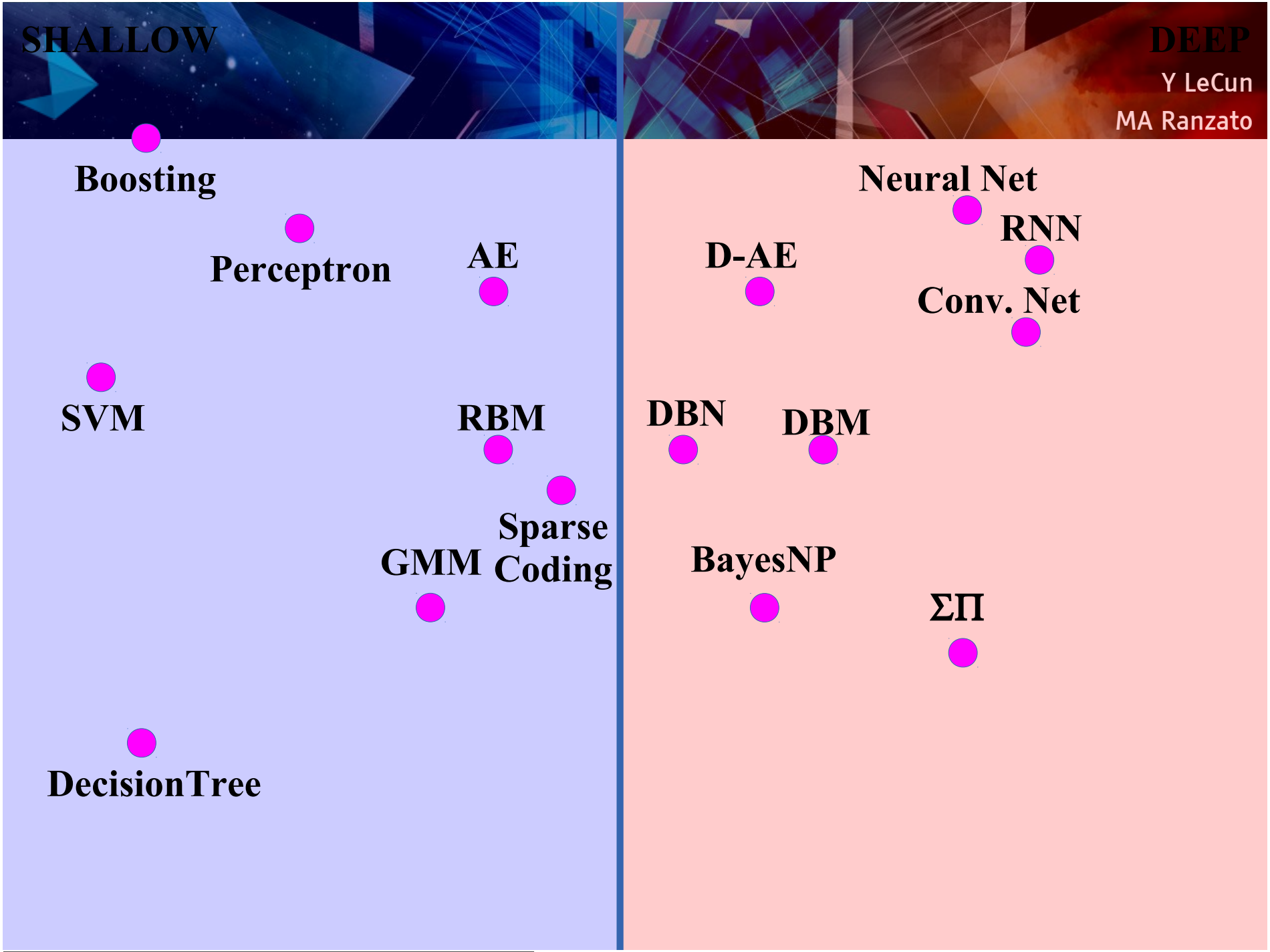
UFMG

Which Models are Deep?

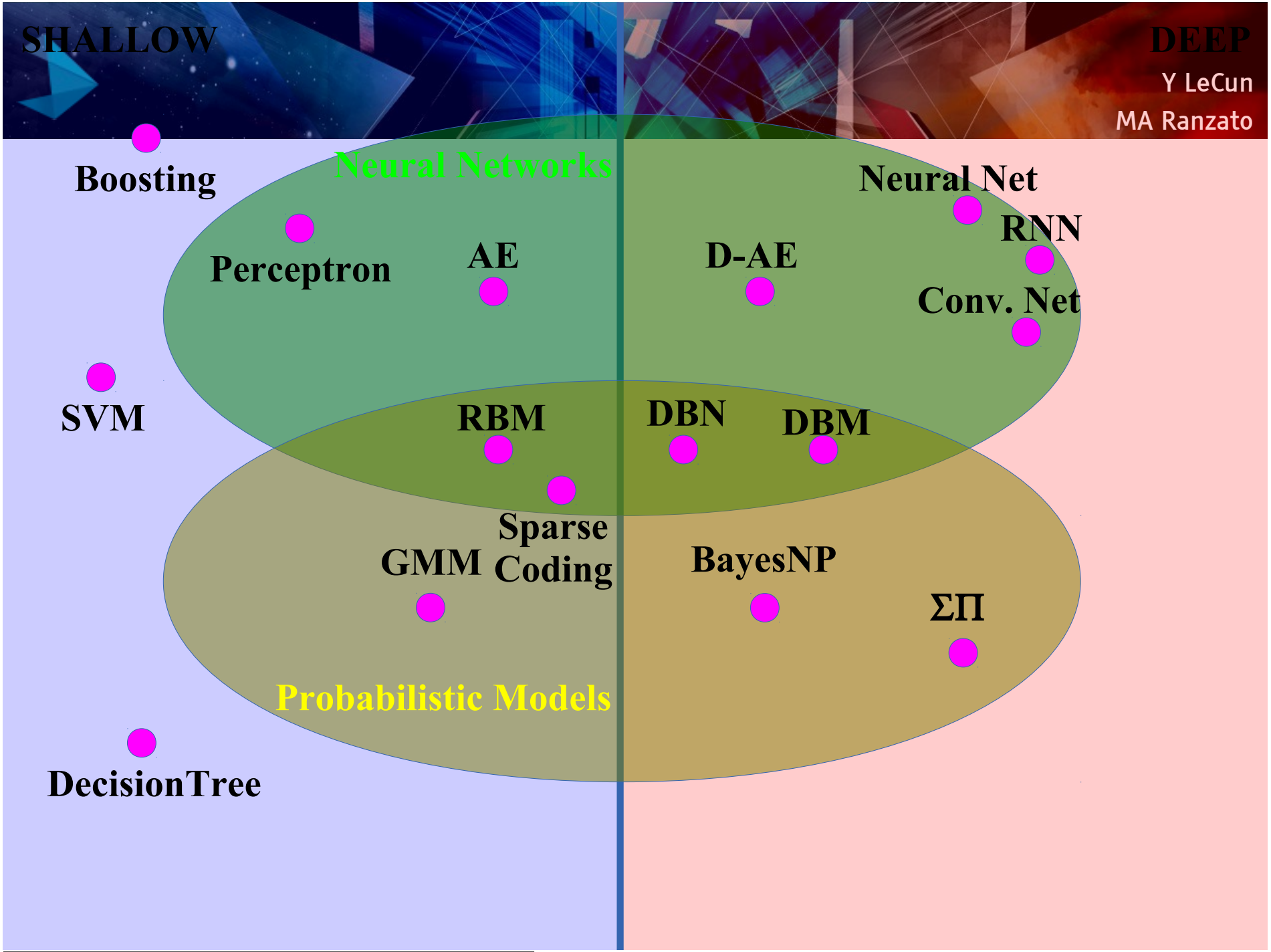
- 2-layer models are not deep (even if you train the first layer)
  - Because there is no feature hierarchy
- Neural nets with 1 hidden layer are not deep
- SVMs and Kernel methods are not deep
  - Layer1: kernels; layer2: linear
  - The first layer is "trained" in with the simplest unsupervised method ever devised: using the samples as templates for the kernel functions.
- Classification trees are not deep
  - No hierarchy of features. All decisions are made in the input space



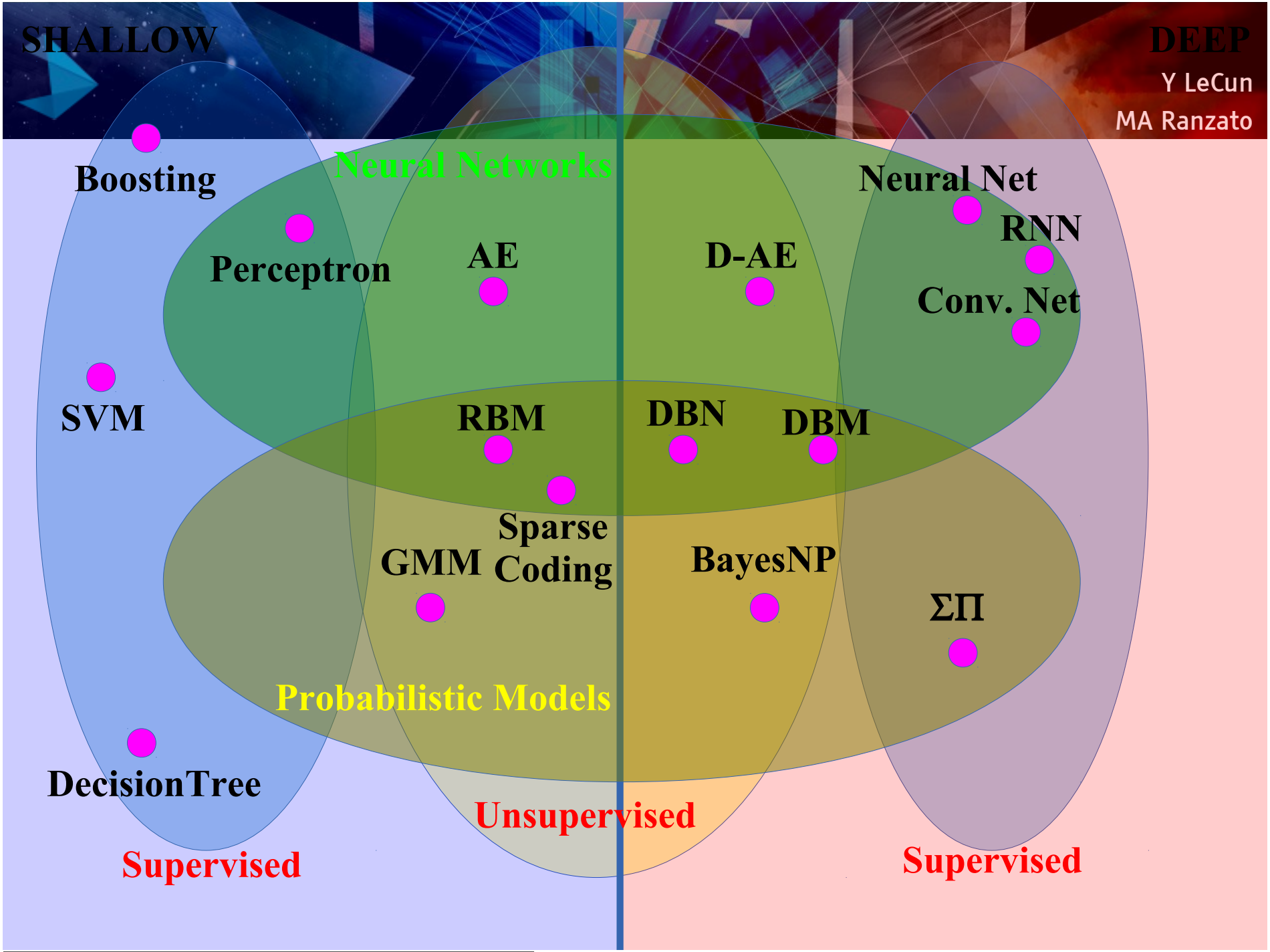
Slide by Prof. Yann Lecun, ICML 2013



Slide by Prof. Yann Lecun, ICML 2013



Slide by Prof. Yann Lecun, ICML 2013



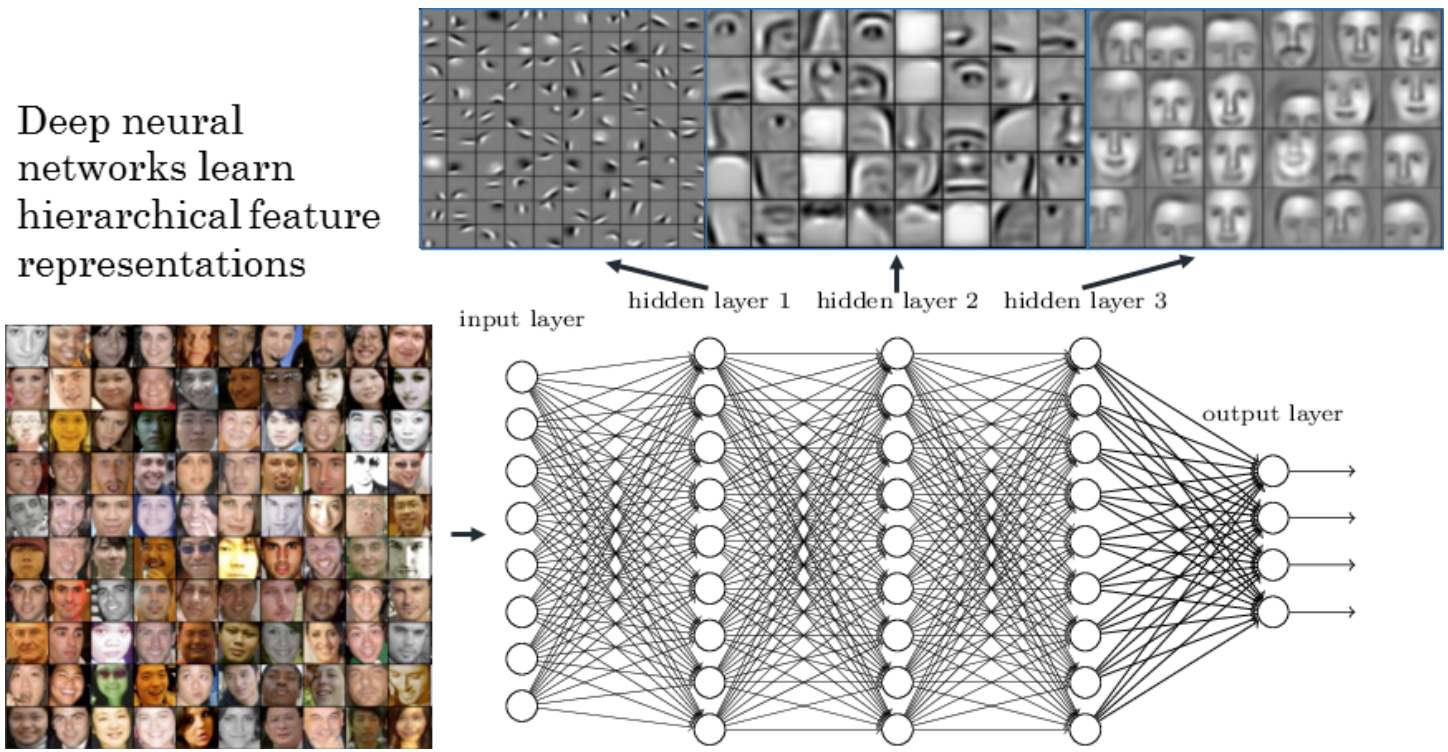
Slide by Prof. Yann Lecun, ICML 2013

Convolutional Neural Networks

UFMG

CNNs, ConvNets

- Can learn specific and adaptable features and classifiers
- Are the most successful strategy in Computer Vision
- Usually obtain different levels of abstraction for the data



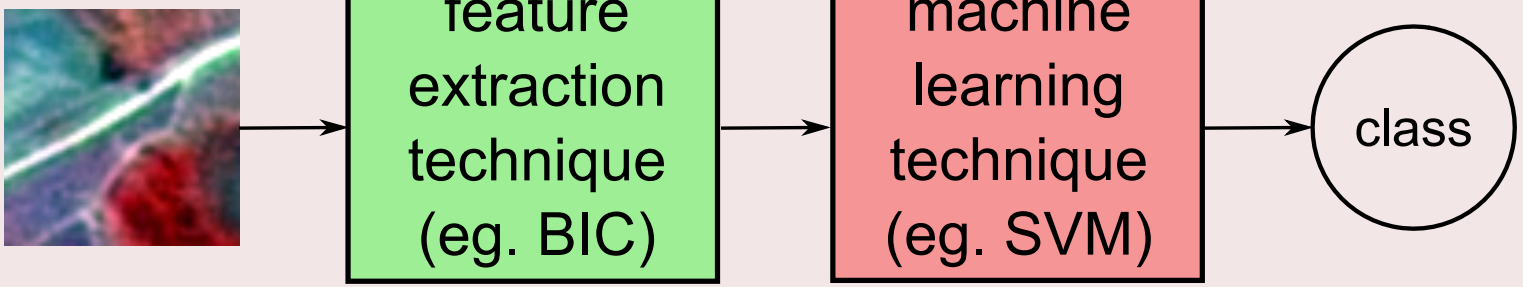
Source: http://www.rsipvision.com/exploring-deep-learning/

Pattern Classifier Approaches

UFMG

Deep vs. Shallow

Shallow

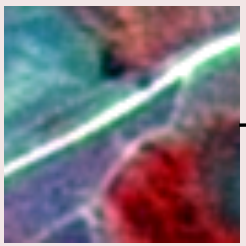


Pattern Classifier Approaches

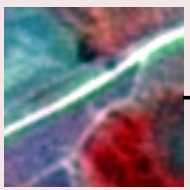
UFMG

Deep vs. Shallow

Shallow



Deep



Convolutional Neural Networks

UFMG

Problems

- It needs too many data
- It is very expensive to compute

Convolutional Neural Networks

UFMG

Problems

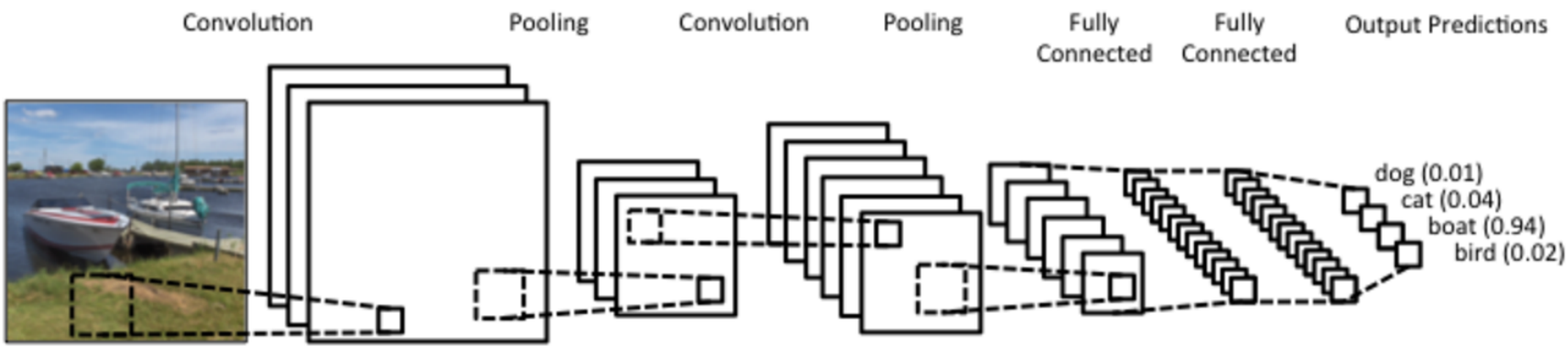
- It needs too many data
- It is very expensive to compute

Solutions

- ImageNET dataset
- Data augmentation
- GPUs

Convolutional Neural Networks

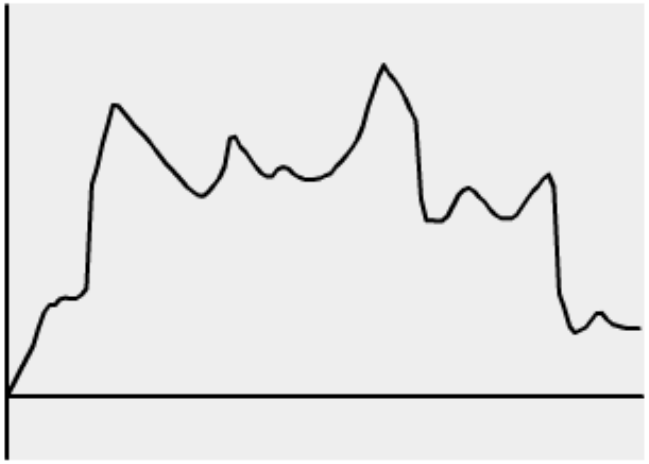
UFMG



Convolution

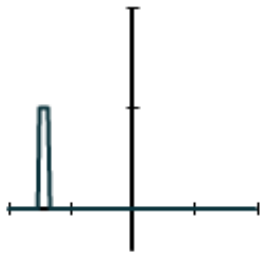
1D signal

(draw to change)



1D filter

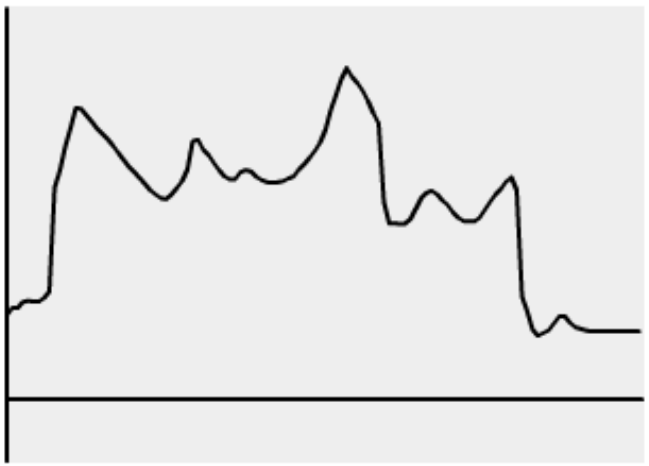
- rect
- big rect
- gaussian
- sharpen
- shift
- custom



animate

normalize

1D result



Help

Reset

2D signal

select image: humming...

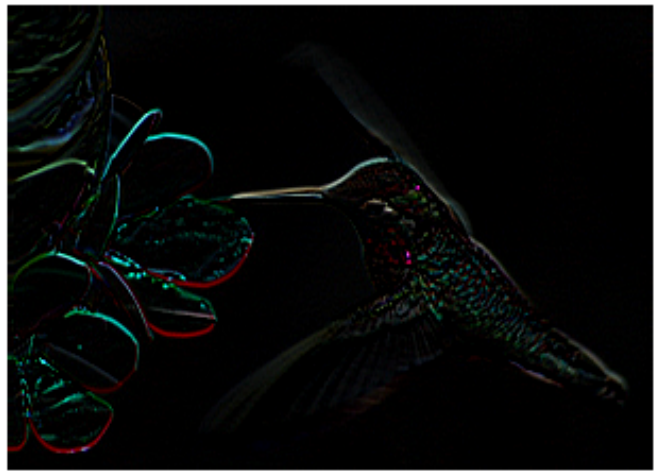
2D filter

- rect
- big rect
- gaussian
- sharpen
- edges
- shift
- hand shake
- custom

0.00	0.00	0.00	0.00	0.00
0.00	0.00	2	0.00	0.00
0.00	-1.00	1	-1.00	0.00
0.00	0.00	-1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00

normalize

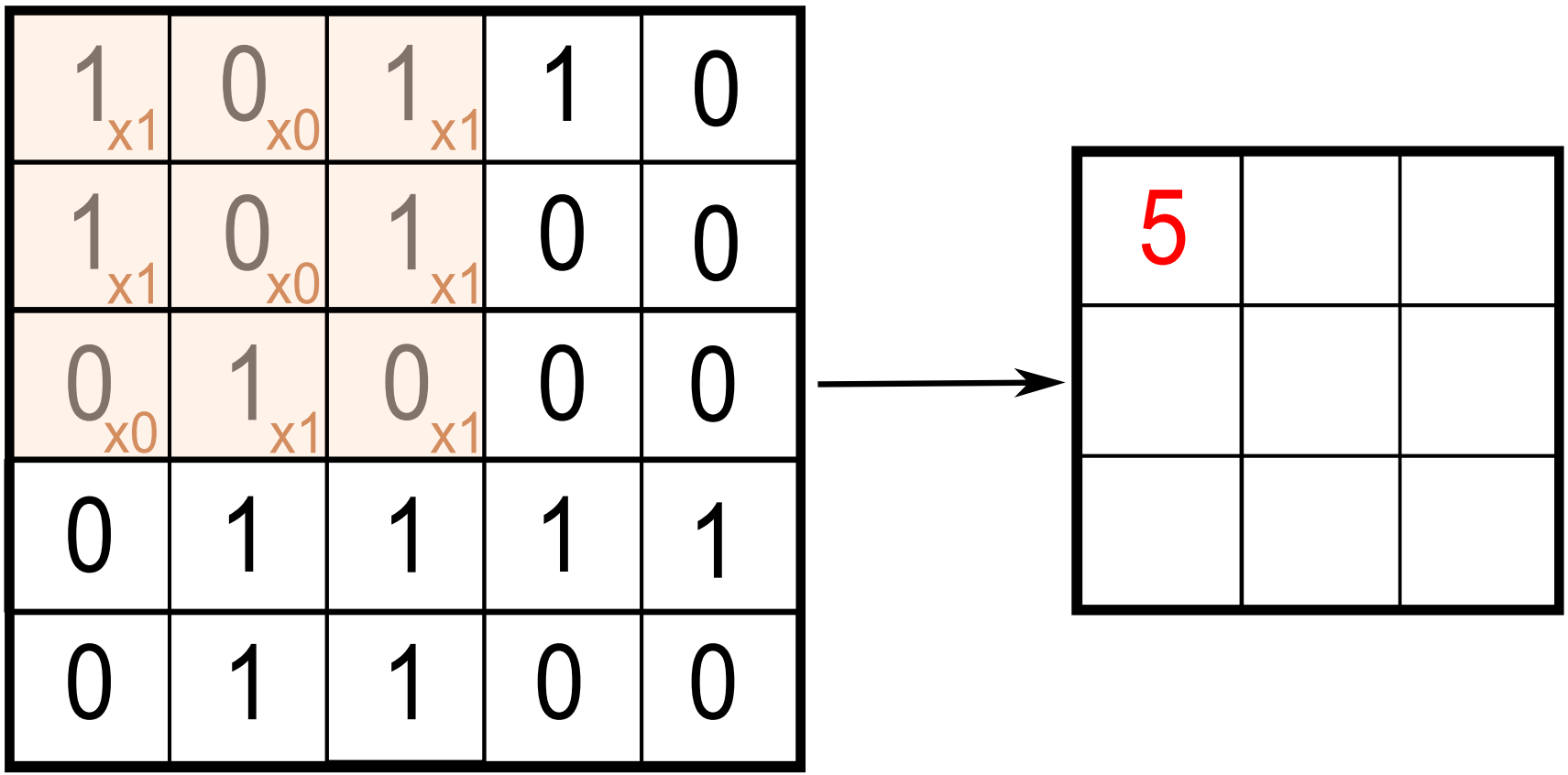
2D result



Convolutional Networks

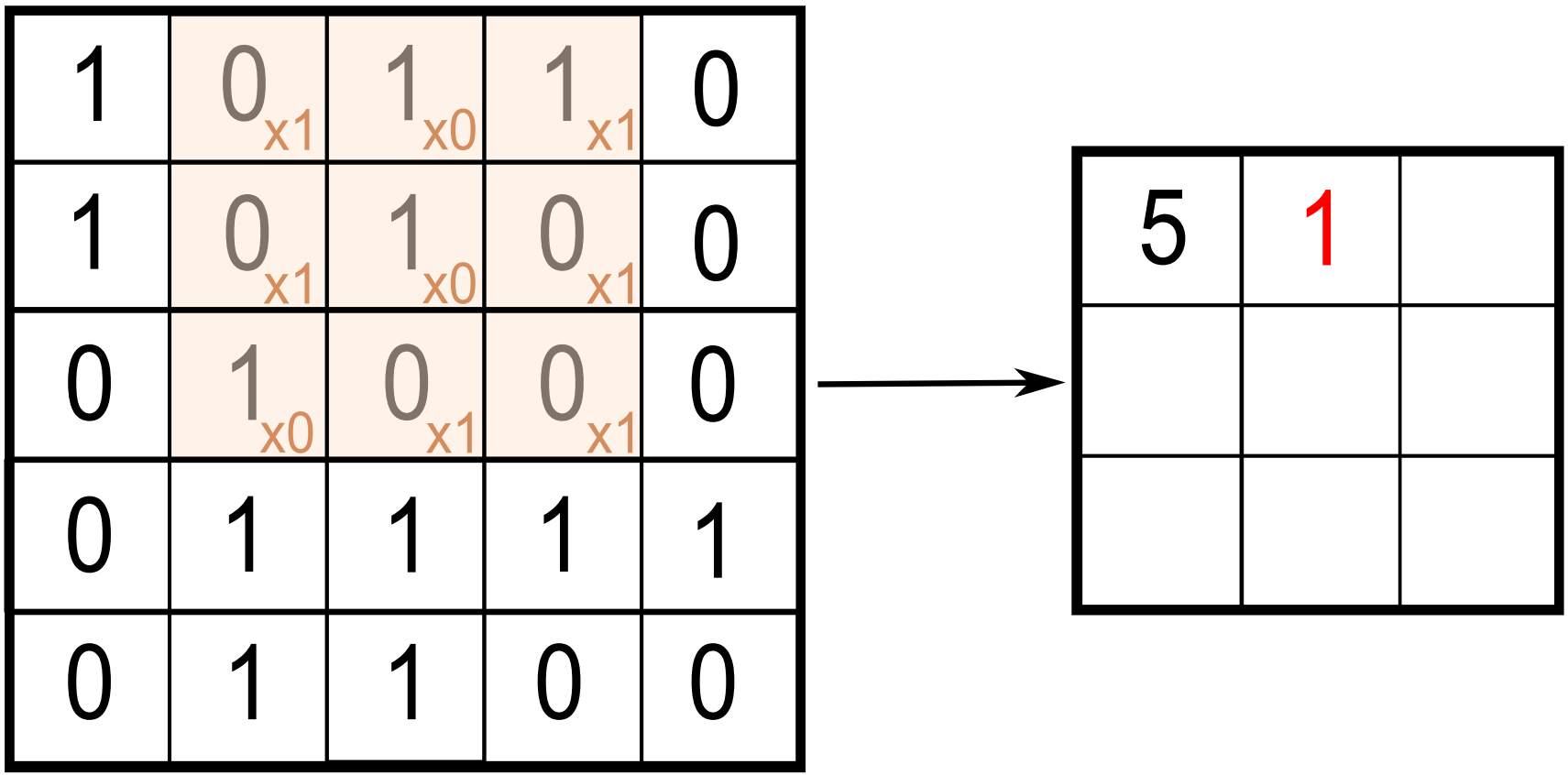
UFMG

Convolutional Layers



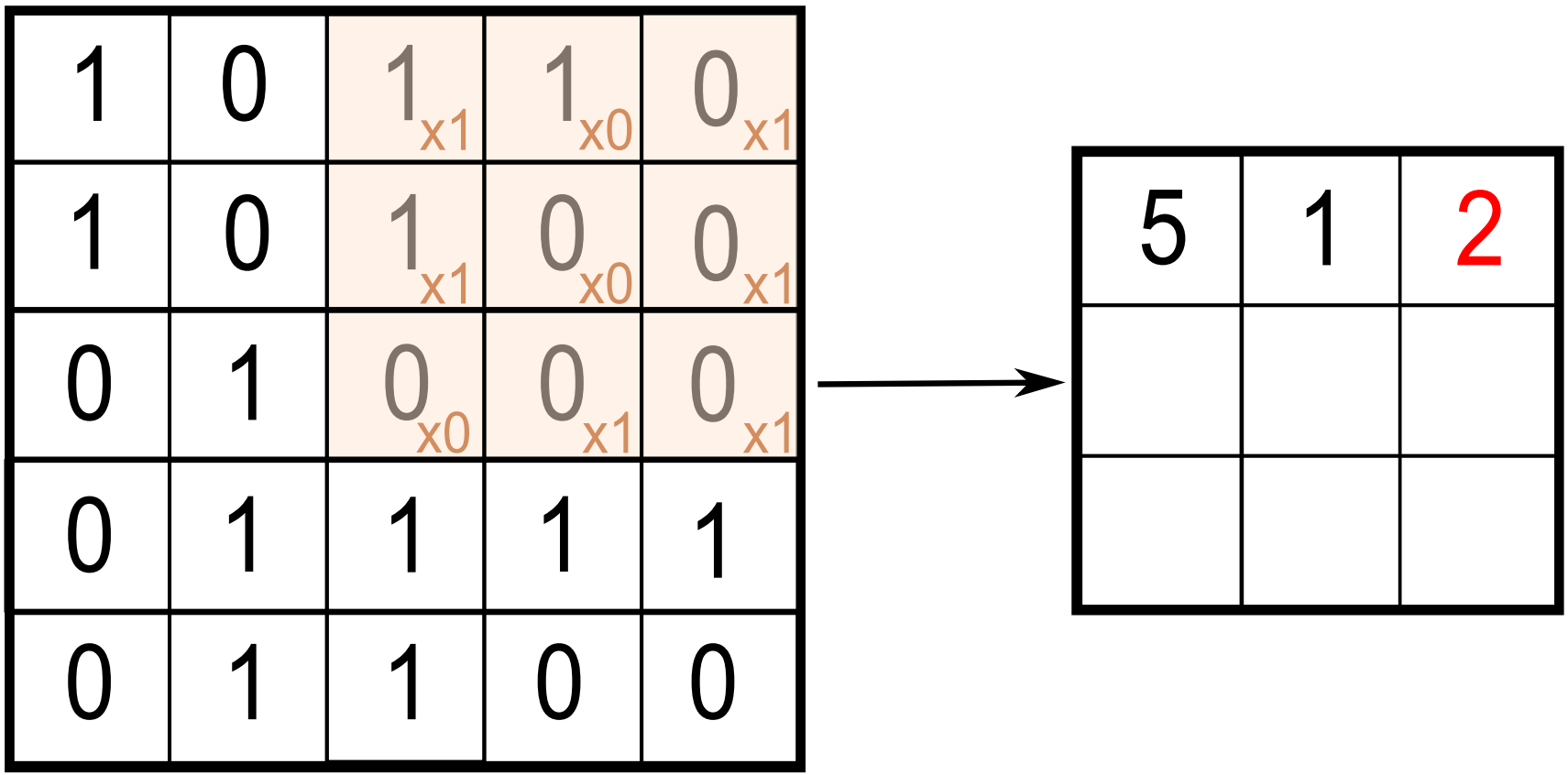
Convolutional Networks

Convolutional Layers



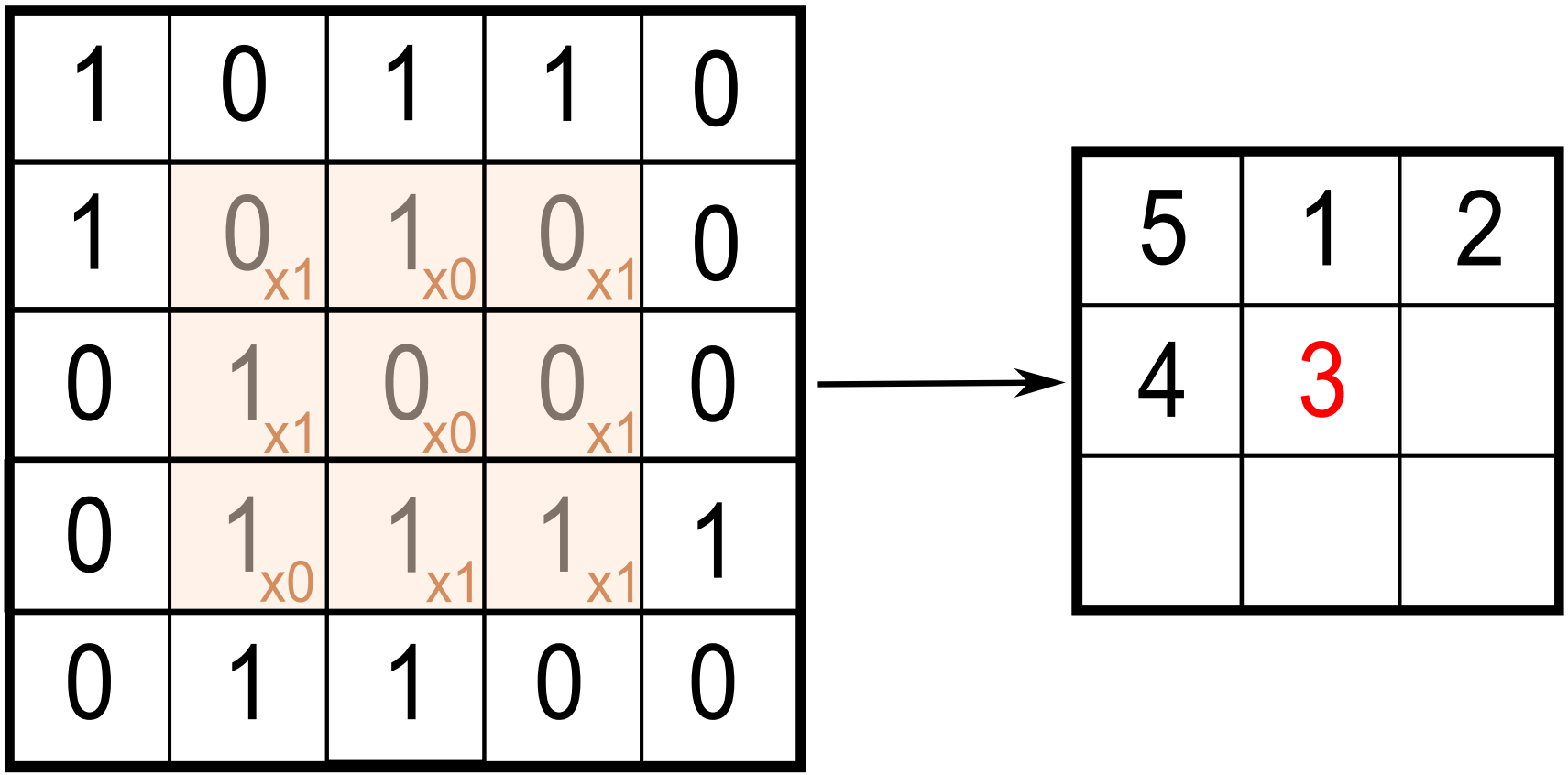
Convolutional Networks

Convolutional Layers



Convolutional Networks

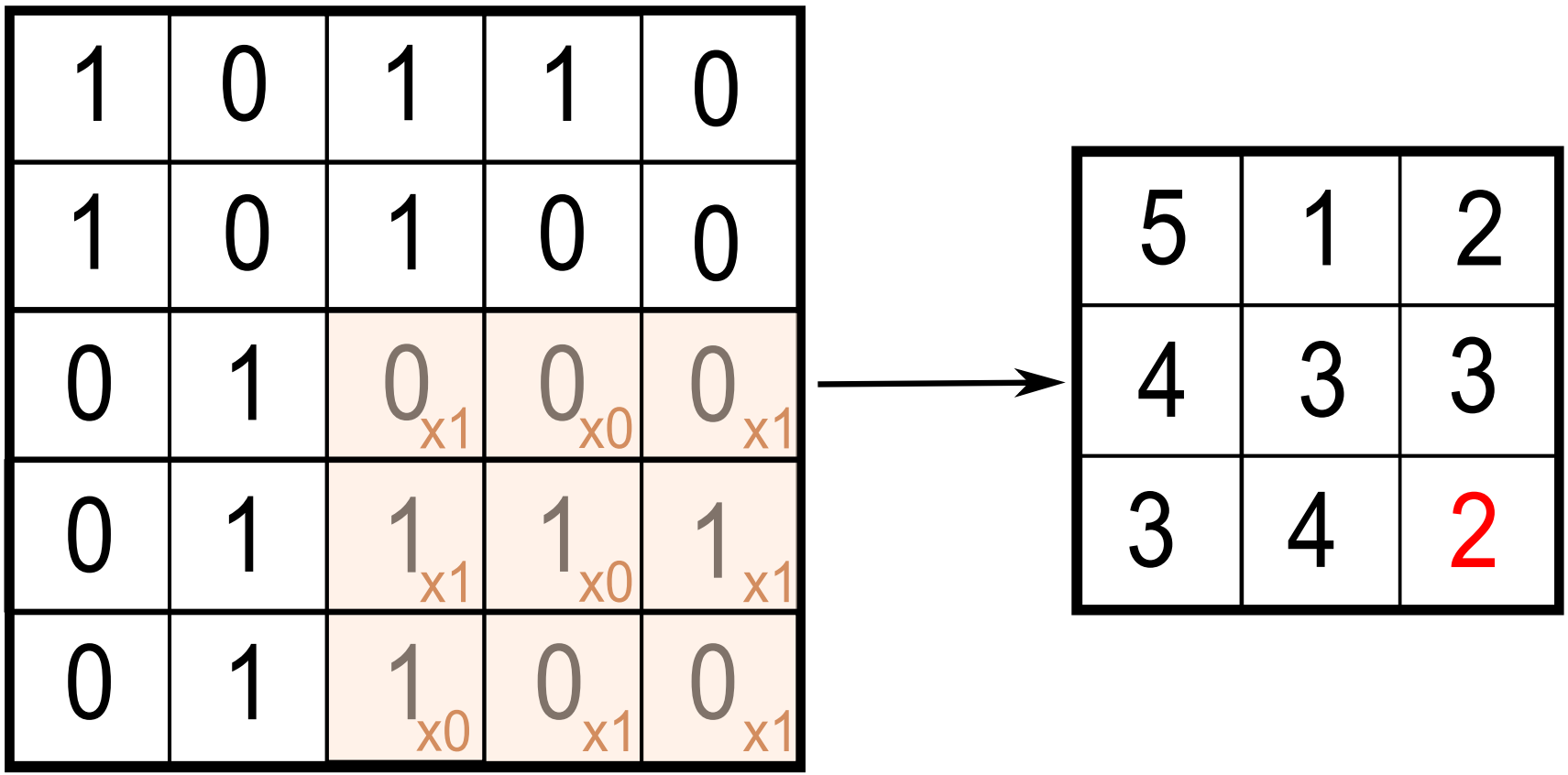
Convolutional Layers



Convolutional Networks

UFMG

Convolutional Layers



Convolutional Networks

UFMG

Pooling Layers

How to use convolved features for classification?

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Networks

UFMG

Pooling Layers

How to use convolved features for classification?

Use all the extracted features

Computationally challenging:

1. Condider images of size 96×96 pixels

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Networks

UFMG

Pooling Layers

How to use convolved features for classification?

Use all the extracted features

Computationally challenging:

1. Condider images of size 96×96 pixels
2. Suppose we have learned 400 features over 8×8 inputs

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Networks

UFMG

Pooling Layers

How to use convolved features for classification?

Use all the extracted features

Computationally challenging:

1. Condider images of size 96×96 pixels
2. Suppose we have learned 400 features over 8×8 inputs
3. This results in a vector of $89^2 * 400 = 3168400$ features

Curse of dimensionality $\Rightarrow$ Overfitting!

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Networks

UFMG

Pooling Layers

How to use convolved features for classification?

Use all the extracted features

Computationally challenging:

1. Condider images of size 96×96 pixels
2. Suppose we have learned 400 features over 8×8 inputs
3. This results in a vector of $89^2 * 400 = 3168400$ features

Curse of dimensionality $\Rightarrow$ Overfitting!

Pooled Features

Idea:

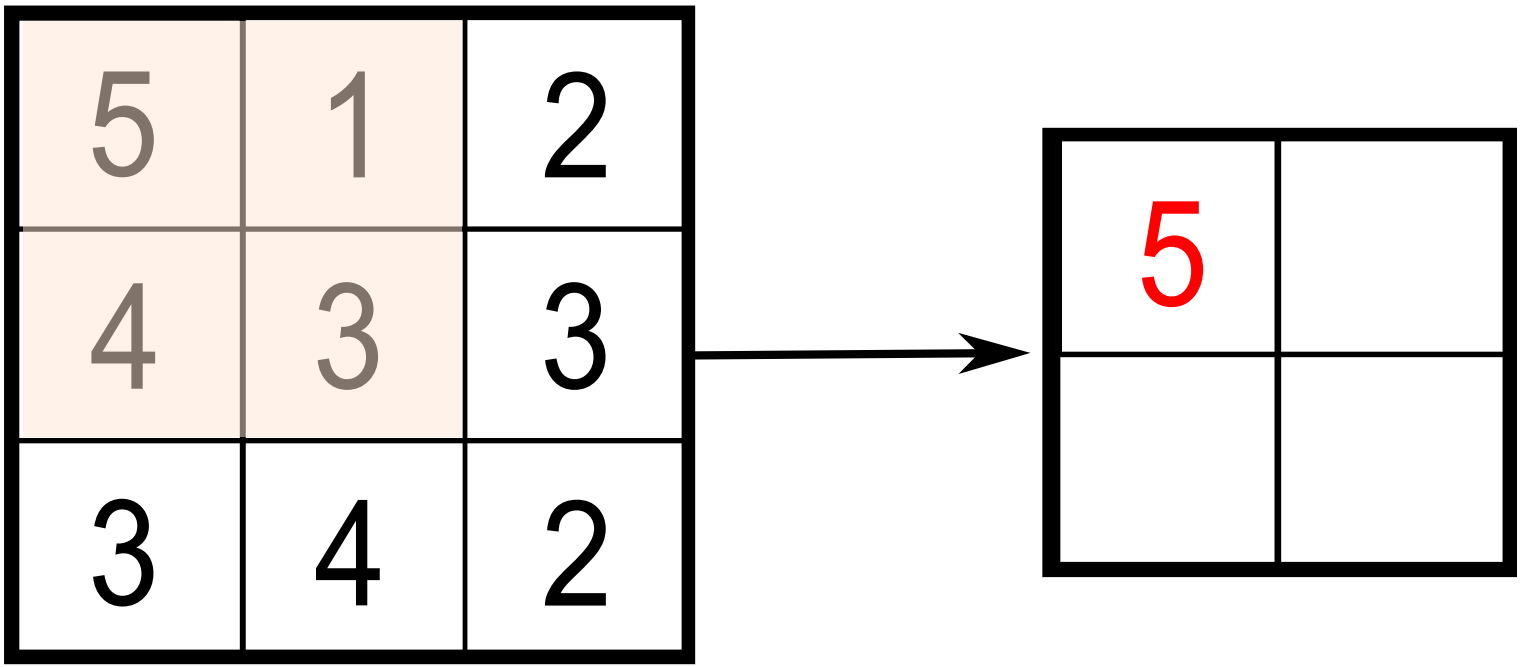
- Aggregate statistics of these features (e.g., mean, max, min)

[1] http://ufldl.stanford.edu/tutorial/supervised/Pooling/

Convolutional Networks

UFMG

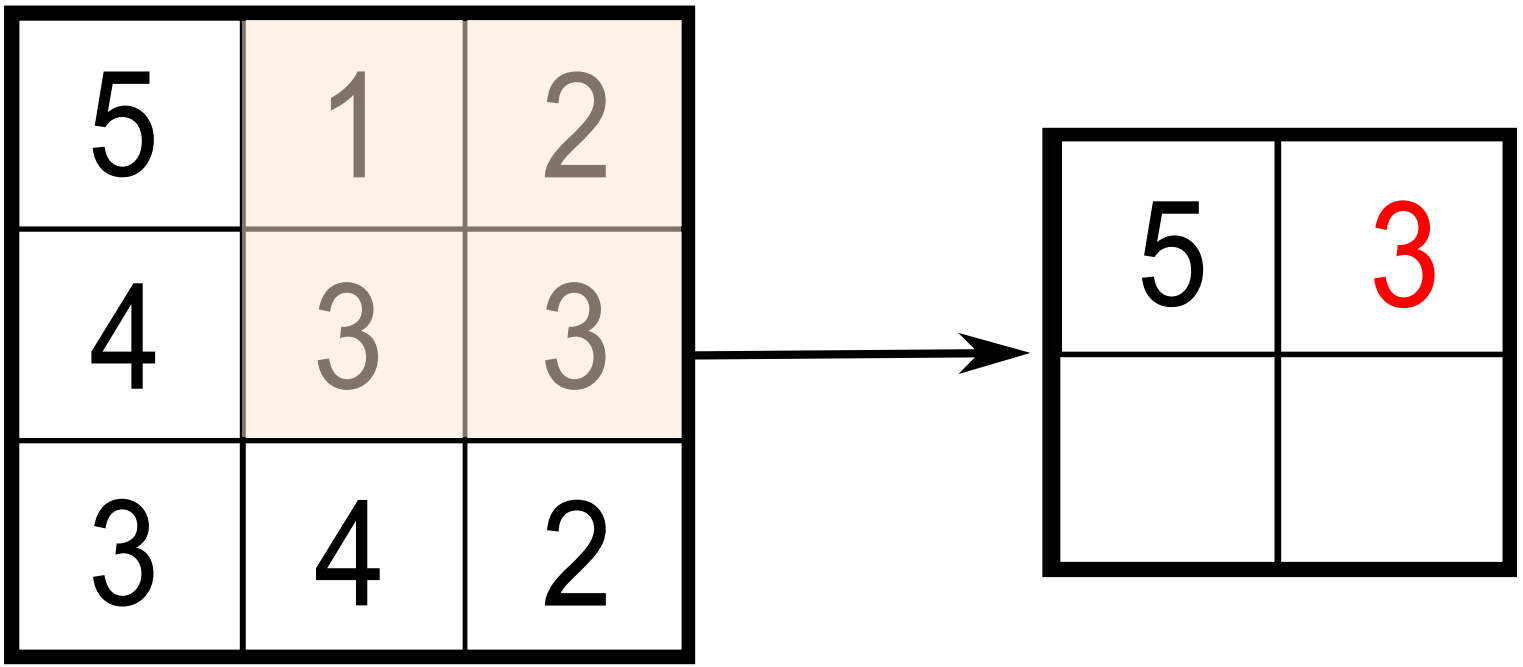
Pooling Layers



Convolutional Networks

UFMG

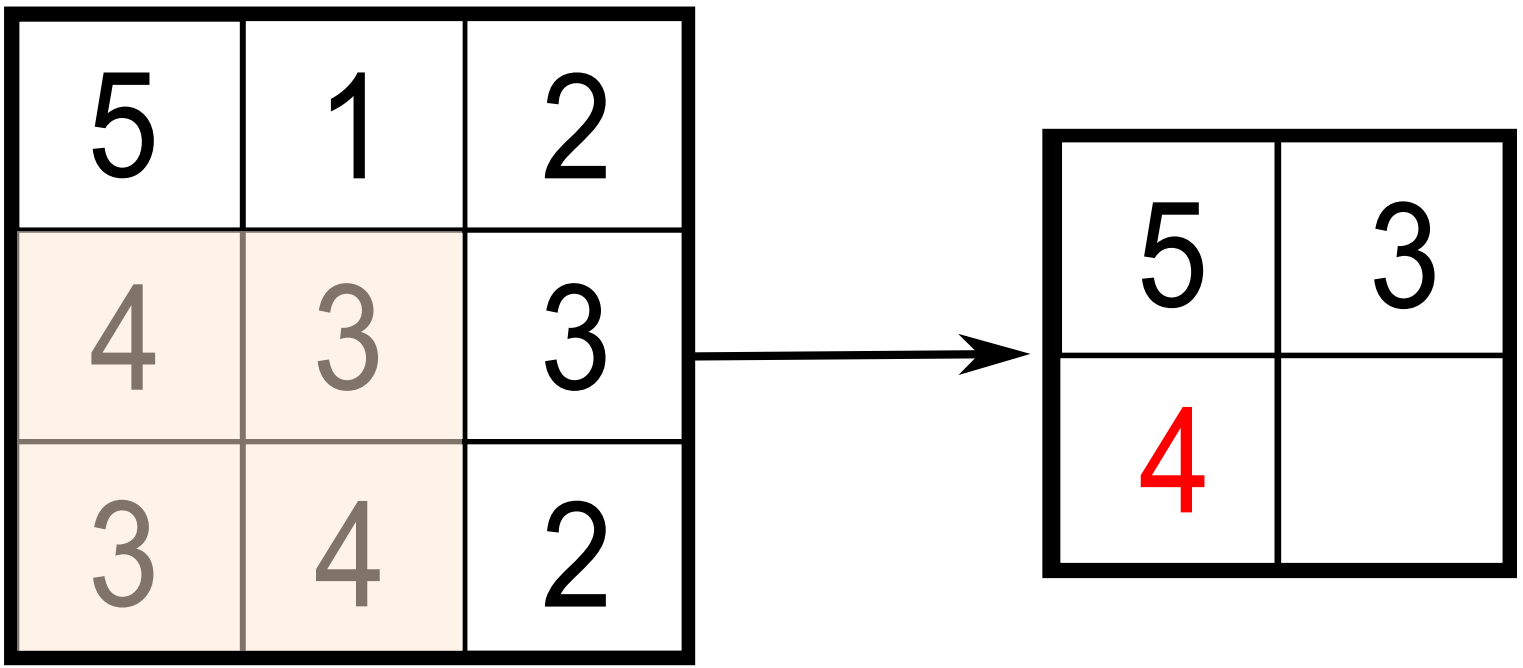
Pooling Layers



Convolutional Networks

UFMG

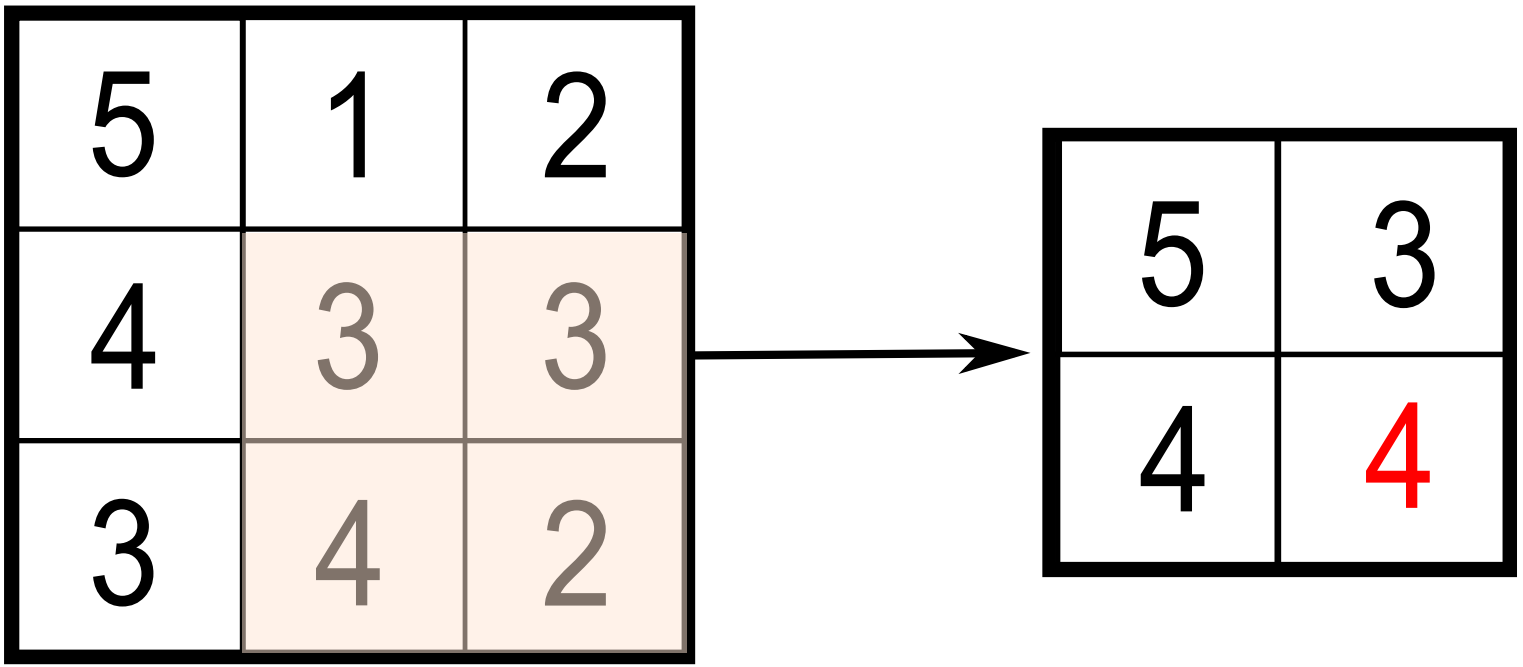
Pooling Layers



Convolutional Networks

UFMG

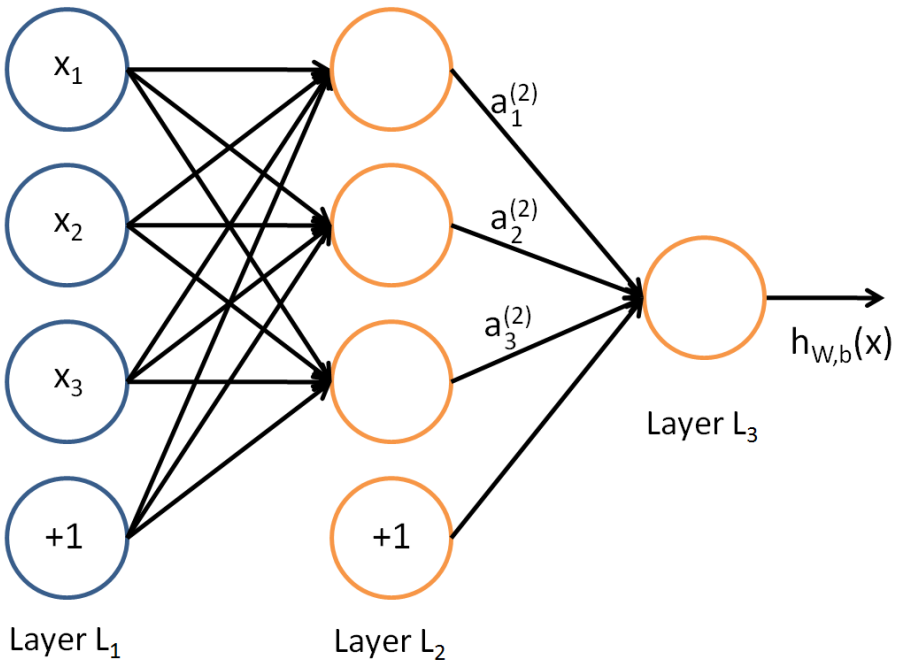
Pooling Layers



Convolutional Networks

UFMG

Fully-Connected Layers



Convolutional Networks

UFMG

Softmax

- Classification layer: calculate the class probability of each instance
  - Softmax, in this case

$$h_{W,b}(X) = P(y = j|X; W, b) = \frac{\exp^{X^T W_j}}{\sum_{k=1}^{K} \exp^{X^T W_k}}$$

Convolutional Networks

UFMG

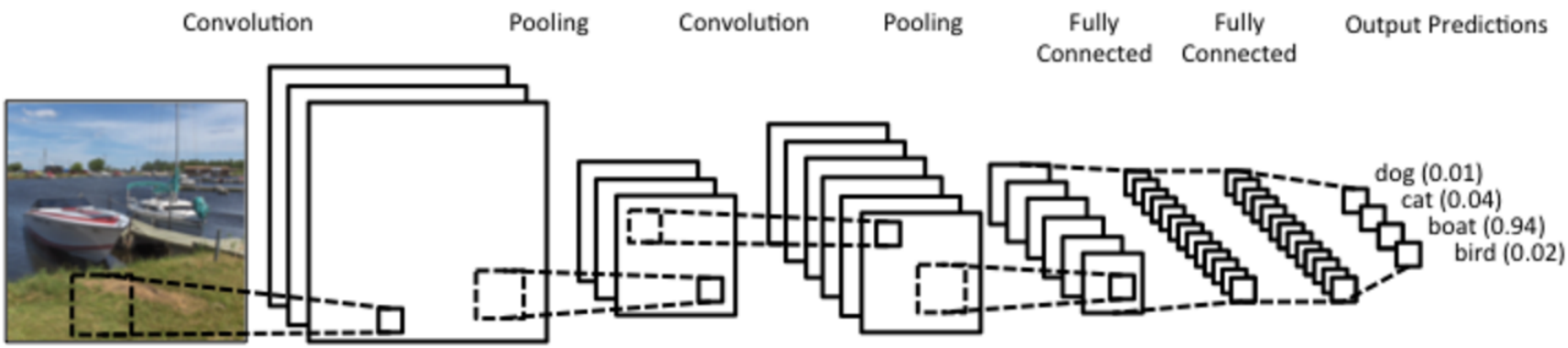
Cost Function

$$\mathscr{J}(W,b) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=0}^{1} 1\{y^{(i)} = k\} \times$$
$$\times P(y^{(i)} = j | x^{(i)}; W, b) + \frac{\lambda}{2}\sum W^2$$

where y represents a possible class
x is the data of an instance
W the weights
i is an specific instance
N represents the total number of instances
$1\{\cdot\}$ is the "indicator function"

Convolutional Neural Networks

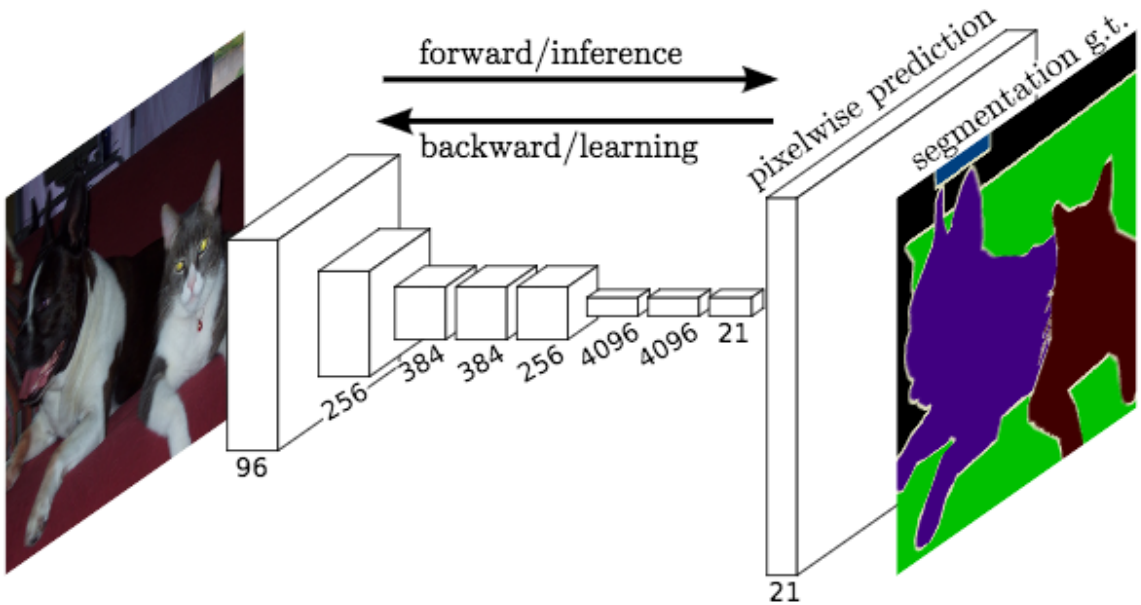
UFMG



Other Convolutional Architectures

UFMG

- **Fully-Convolutional Networks**
- Convolutional Auto-Encoders
- Deconvolution Networks
- Dilated Convolucional Networks

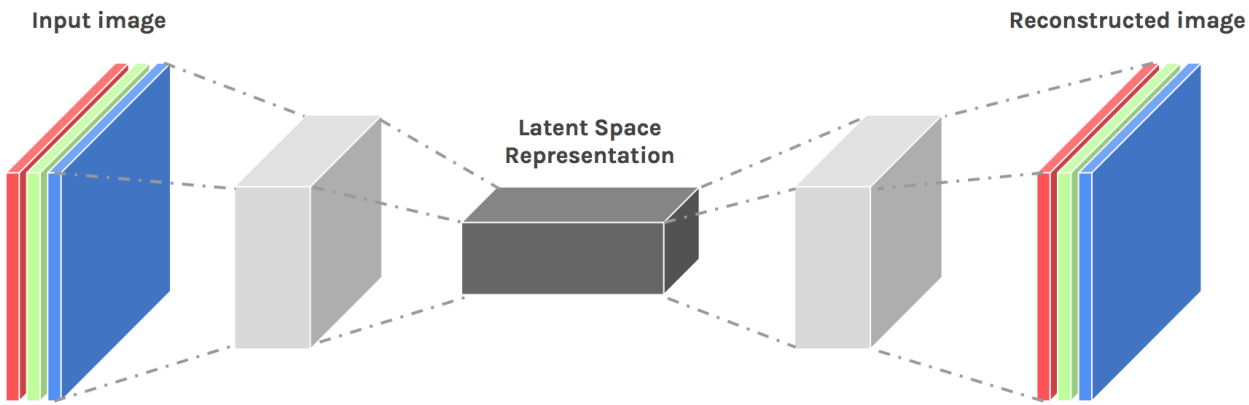


Application: semantic segmentation

Other Convolutional Architectures

UFMG

- Fully-Convolutional Networks
- **Convolutional Auto-Encoders**
- Deconvolution Networks
- Dilated Convolucional Networks

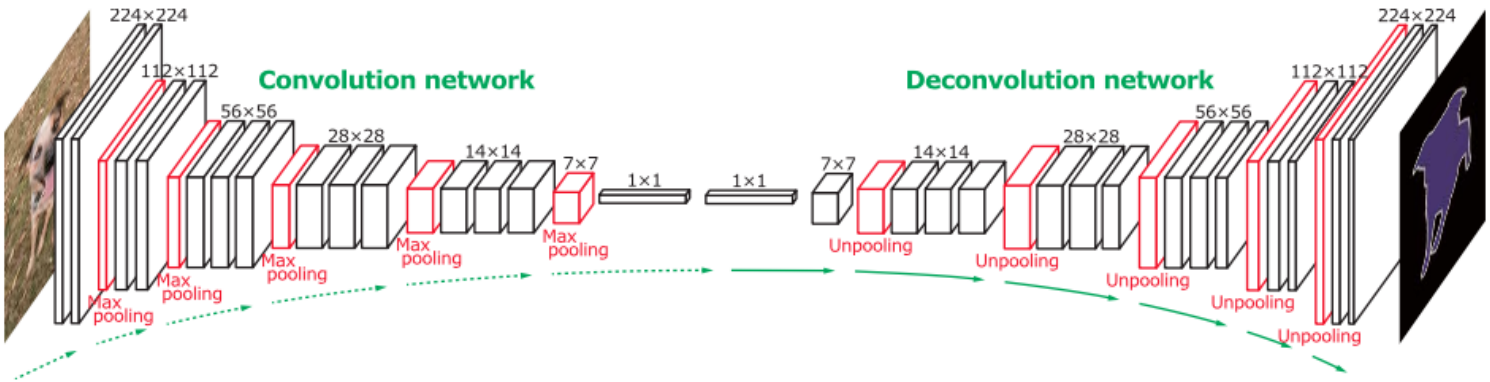


Applications: compaction, denoising, super-resolution

Other Convolutional Architectures

UFMG

- Fully-Convolutional Networks
- Convolutional Auto-Encoders
- **Deconvolution Networks**
- Dilated Convolucional Networks

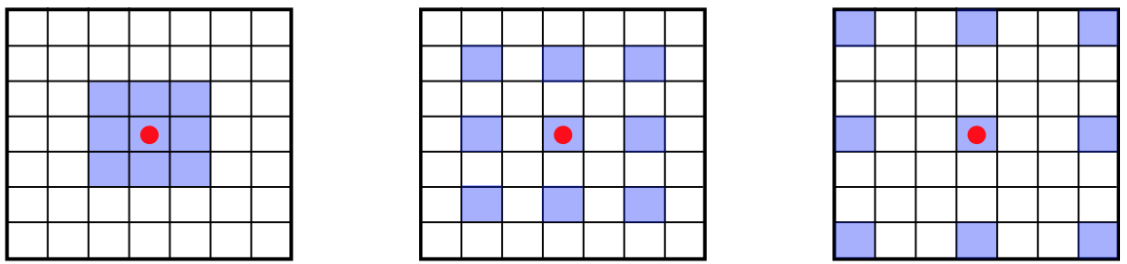


Applications: compaction, denoising, super-resolution, semantic segmentation

Other Convolutional Architectures

UFMG

- Fully-Convolutional Networks
- Convolutional Auto-Encoders
- Deconvolution Networks
- **Dilated Convolucional Networks**



(a) Rate 1 (b) Rate 2 (c) Rate 3

Fig. 1: Example of dilated convolutions. Dilation supports expansion of the receptive field without loss of resolution or coverage of the input.

Applications: compaction, denoising, super-resolution, semantic segmentation

Other Convolutional Architectures

UFMG

- Fully-Convolutional Networks
- Convolutional Auto-Encoders
- Deconvolution Networks
- **Dilated Convolucional Networks**