Machine Learning

4771

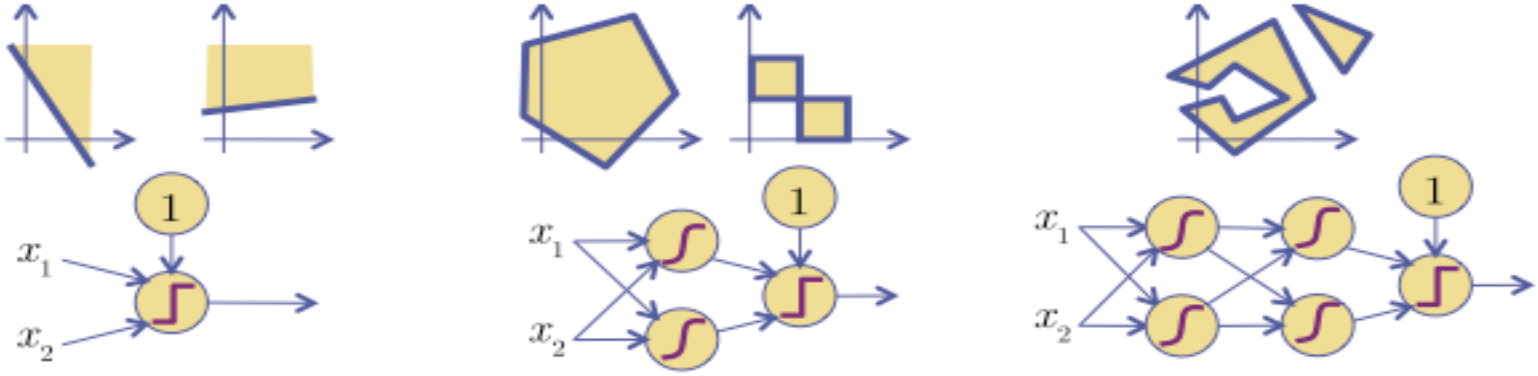
Instructors:
Adrian Weller and Ilia Vovsha

Lecture 7: Neural Networks & Back-Propagation

- Neural Networks (Bishop 5.1-5.3.2)
- Network Learning, Lagrange multipliers
- Back-Propagation

Multi-Layer Neural Network (idea)

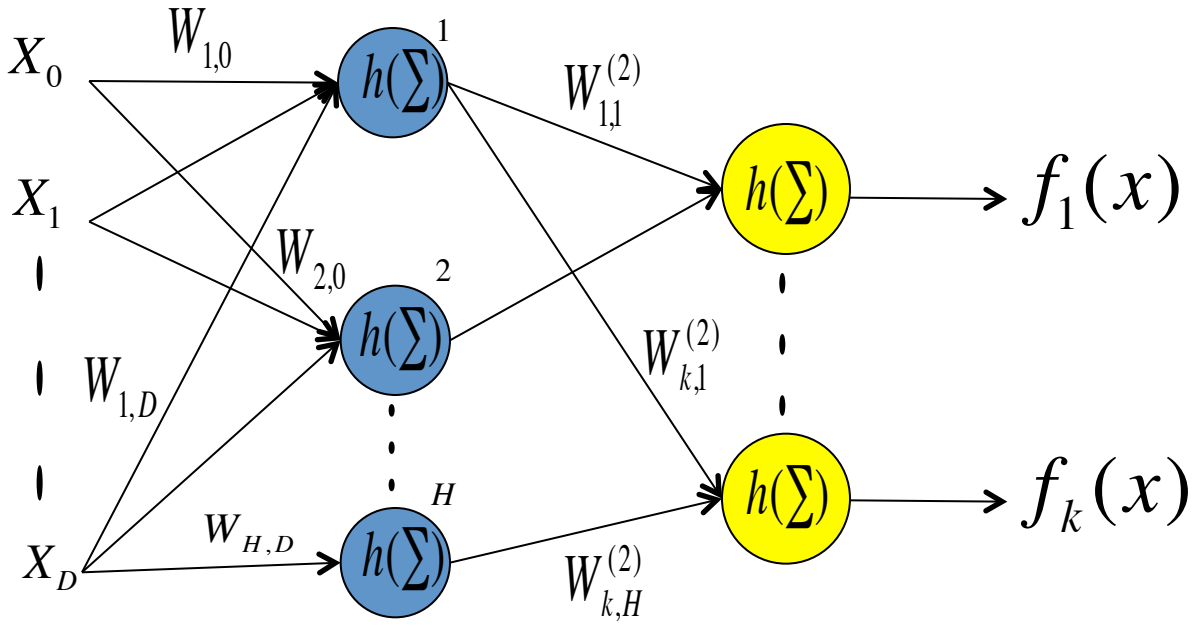
- 1-layer (perceptron): can't even handle XOR!
- What if we consider cascading multiple layers of network?
- Each output layer is input to the next layer
- Each layer has its own weights (parameters)
- Each layer adds more flexibility (but more parameters!)
- Each node splits its input space with linear hyperplane



- Multi-Layer Network can handle more complex decisions
- Note: Without loss of generality, we can use augmented vectors

Formal Definition

- Two-layer feed-forward neural network:
  - Edges (synapses): multiply signal by scalar weight
  - Nodes: sum the inputs
  - Layers: 1. input vector 2. hidden units 3. output unit(s)
  - Parameters: $\{w_{1,0} \ldots w_{H,D}\}$ $\{w_{1,1}^{(2)} \ldots w_{k,H}^{(2)}\}$
  - Activation functions: differentiable & nonlinear (sigmoidal)



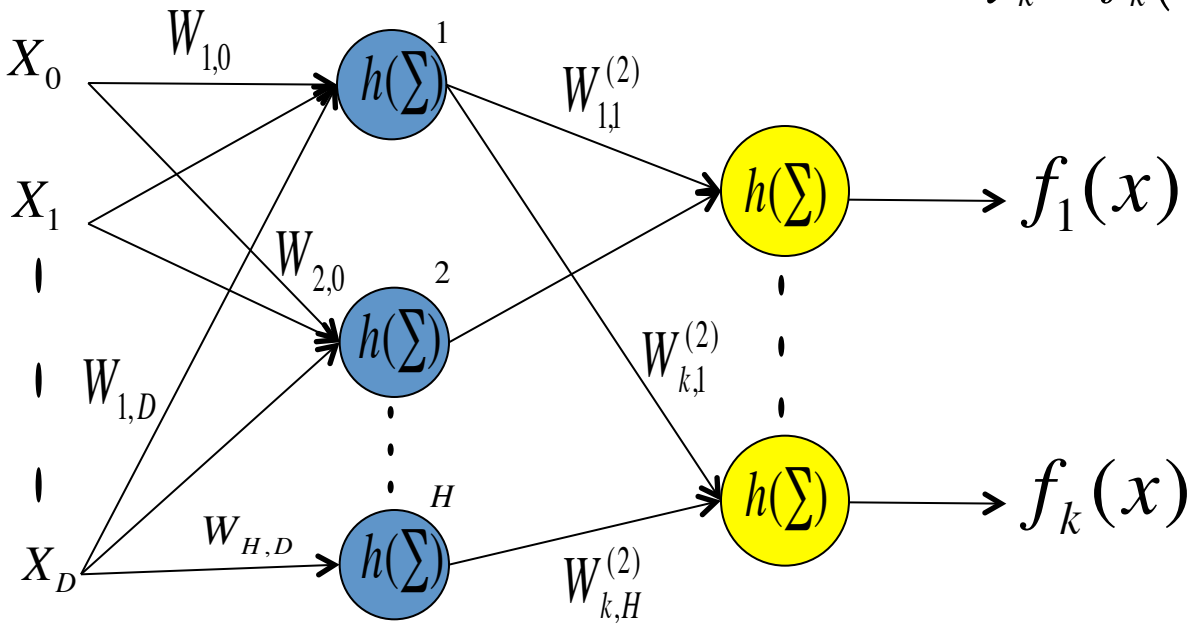
Formal Definition

$$a_j = \sum_{i=1}^{D} w_{ji} x_i$$

$$z_j = h\left(a_j\right)$$

$$a_k^{(2)} = \sum_{j=1}^{H} w_{kj}^{(2)} z_j$$

$$y_k = f_k\left(x\right) = h\left(a_k^{(2)}\right)$$



Expressive Power of NNs

- Discriminant function is expressed as:

$$f_k(x) = h\left(\sum_{j=1}^{H} w_{kj}\, h\left(\sum_{i=1}^{D} w_{ji} x_i\right)\right)$$

- Can any (continuous) function be implemented by a 2-layer (1 hidden layer) net?
  - Yes! (with enough hidden units; result due to Kolmogorov)
  - Valuable result theoretically, not practically (why?)

Network Learning Problem

- Cost function to minimize: $R(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} E(y_i, t_i)$
- Subject to conditions: $z_j = h\left(\sum_{i=1}^{D} w_{ji} x_i\right), \quad j = 1,\ldots,H$

$$y_k = h\left(\sum_{j=1}^{H} w_{kj}^{(2)} z_j\right), \quad k = 1,\ldots,K$$

$$a_j = \sum_{i=1}^{D} w_{ji} x_i$$

$$z_j = h\left(a_j\right)$$

$$a_k^{(2)} = \sum_{j=1}^{H} w_{kj}^{(2)} z_j$$

$$y_k = f_k\left(x\right) = h\left(a_k^{(2)}\right)$$

Network Learning Problem

- Cost function to minimize: $R(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} E(y_i, t_i)$
- Subject to conditions:

$$z_j = h\left(\sum_{i=1}^{D} w_{ji} x_i\right), \quad j = 1,\ldots,H$$

$$y_k = h\left(\sum_{j=1}^{H} w_{kj}^{(2)} z_j\right), \quad k = 1,\ldots,K$$

- Optimization problem (one output, one input):

$$\min R(\mathbf{w})$$

$$s.t. \quad z_j - h\left(\sum_{i=1}^{D} w_{ji} x_i\right) = 0, \quad j = 1,\ldots,H$$

$$s.t. \quad y - h\left(\sum_{j=1}^{H} w_{1,j}^{(2)} z_j\right) = 0$$

Optimization: Equality Constraints

- Problem: given a function of several variables, find its stationary point subject to one constraint
- Formally (two-variable case):

$$\max_{x_1,\, x_2} \; f(x_1,\, x_2)$$

$$subject\ to:\; g(x_1,\, x_2) = 0$$

- Formally (general case):

$$\max_{\mathbf{x}} \; f(\mathbf{x})$$

$$s.t. \quad g(\mathbf{x}) = 0$$

Trivial Approach

a. Use $g(x_1 , x_2)$ to express the x_2 in terms of x_1: $x_2 = h(x_1)$

b. Substitute $x_2 = h(x_1)$ into $f(x_1 , x_2)$, and differentiate to find optimal (stationary) value x_1^*

c. Substitute back to get $x_2^* = h(x_1^*)$

- Why not?

Better Approach (Geometry)

• Consider the geometry of the problem:

a. Assume x is D-dimensional. Then the constraint is a D-1 dimensional surface in x-space

b. At any point on the constraint surface $\nabla g(\mathbf{x})$ is orthogonal to the surface

Why?

c. If f(x) is maximized at point x^* on the constraint surface, $\nabla f(\mathbf{x})$ is orthogonal to the constraint surface at x^*

Why?

d. There must exist a parameter λ s.t. $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0, \quad \lambda \neq 0$

Why?

Better Approach (Geometry)

• Consider the geometry of the problem:

a. Assume x is D-dimensional. Then the constraint is a D-1 dimensional surface in x-space

b. At any point on the constraint surface $\nabla g(\mathbf{x})$ is orthogonal to the surface

Consider two points {x, x+ε} on g, and make a Taylor series expansion around x:

$$g(\mathbf{x}+\mathbf{e}) \approx g(\mathbf{x}) + \mathbf{e}^T \nabla g(\mathbf{x})$$

$$g(\mathbf{x}+\mathbf{e}) = g(\mathbf{x}) \Rightarrow \lim_{\|\mathbf{e}\| \to 0} \mathbf{e}^T \nabla g(\mathbf{x}) = 0$$

$$\mathbf{e} \parallel \{g(\mathbf{x}) = 0\} \Rightarrow \{g(\mathbf{x}) = 0\} \perp \nabla g(\mathbf{x})$$

c. If f(x) is maximized at point x^* on the constraint surface, $\nabla f(\mathbf{x})$ is orthogonal to the constraint surface at x^*

d. There must exist a parameter λ s.t. $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0, \quad \lambda \neq 0$

Better Approach (Geometry)

• Consider the geometry of the problem:

a. Assume x is D-dimensional. Then the constraint is a D-1 dimensional surface in x-space

b. At any point on the constraint surface $\nabla g(\mathbf{x})$ is orthogonal to the surface

Consider two points {x, x+e} on g, and make a Taylor series expansion around x:

$$\lim_{\|\mathbf{e}\|\to 0} \mathbf{e}^T \nabla g(\mathbf{x}) = 0 \Rightarrow \{g(\mathbf{x}) = 0\} \perp \nabla g(\mathbf{x})$$

c. If f(x) is maximized at point x^* on the constraint surface, $\nabla f(\mathbf{x})$ is orthogonal to the constraint surface at x^*

Otherwise we can increase the value of f(x) by moving along the constraint surface

d. There must exist a parameter λ s.t. $\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0, \quad \lambda \neq 0$

$\nabla f(\mathbf{x}), \nabla g(\mathbf{x})$ are (anti) parallel

Lagrange Multipliers

A. Define a function: $L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x}), \quad \lambda \neq 0$

   - ***L*** is the Lagrangian, λ is the Lagrange Multiplier

B. Find the stationary point of ***L*** with respect to {x, λ}:

$$\nabla_x L = 0, \quad \frac{\partial L}{\partial \lambda} = 0$$

$$\nabla f(\mathbf{x}) + \lambda \nabla g(\mathbf{x}) = 0, \quad g(\mathbf{x}) = 0$$

C. Obtain D+1 equations

   - D equations to determine $\mathbf{x}^*$ (one equation for λ)
   - Might not care about λ, hence can be eliminated (undetermined multiplier)

Multiple Equality Constraints

• Problem: given a function of several variables, find its stationary point subject to one or more constraints

• Formally (general case):

$$\max_{\mathbf{x}} \; f(\mathbf{x})$$

$$s.t. \quad g_j(\mathbf{x}) = 0, \quad j = 1,\ldots,J$$

•Define the Lagrangian:

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \sum_{j=1}^{J} \lambda_j g_j(\mathbf{x}), \quad \forall j : \lambda_j \neq 0$$

Network Learning Solution

- Problem: $\min R(\mathbf{w})$

$$s.t. \quad z_j - h\left(\sum_{i=1}^{D} w_{ji} x_i\right) = 0, \quad j = 1,\ldots,H$$

$$s.t. \quad y - h\left(\sum_{j=1}^{H} w_{1,j}^{(2)} z_j\right) = 0$$

- Lagrangian:

$$L(\mathbf{w},\mathbf{z},\boldsymbol{\lambda}) = E(z_{H+1}, t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h\left(\sum_{i=1}^{D} w_{ji} x_i\right)\right] + \lambda_{H+1}\left[z_{H+1} - h\left(\sum_{j=1}^{H} w_j^{(2)} z_j\right)\right]$$

$$L(\mathbf{w},\mathbf{z},\boldsymbol{\lambda}) = E(z_{H+1}, t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h\left(a_j\right)\right] + \lambda_{H+1}\left[z_{H+1} - h\left(a_{H+1}\right)\right]$$

Network Learning Solution

- Lagrangian:

$$L(\mathbf{w},\mathbf{z},\boldsymbol{\lambda}) = E(z_{H+1},t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h\left(\sum_{i=1}^{D} w_{ji} x_i \right) \right] + \lambda_{H+1} \left[z_{H+1} - h\left(\sum_{j=1}^{H} w_j^{(2)} z_j \right) \right]$$

$$L(\mathbf{w},\mathbf{z},\boldsymbol{\lambda}) = E(z_{H+1},t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h\left(a_j \right) \right] + \lambda_{H+1} \left[z_{H+1} - h\left(a_{H+1} \right) \right]$$

- Gradient:

$$\forall j : \frac{\partial L}{\partial \lambda_j} = 0, \quad \forall j : \frac{\partial L}{\partial z_j} = 0, \quad \forall i,j : \frac{\partial L}{\partial w_{ij}} = 0$$

Solution (1st subcondition)

- Lagrangian:

$$L(\mathbf{w},\mathbf{z},\boldsymbol{\lambda}) = E(z_{H+1},t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h(a_j) \right] + \lambda_{H+1} \left[z_{H+1} - h(a_{H+1}) \right]$$

- 1st subcondition:

$$\forall j : \frac{\partial L}{\partial \lambda_j} = 0 \quad \Rightarrow \quad \left\{ \begin{array}{l} \forall j : \quad z_j - h(a_j) = 0 \\ z_{H+1} - h(a_{H+1}) = 0 \end{array} \right\}$$

$$Forward\ dynamics : \left\{ \begin{array}{l} \forall j, \quad z_j = h\left(\sum_{i=1}^{D} w_{ji} x_i \right) \\ z_{H+1} = h\left(\sum_{j=1}^{H} w_j^{(2)} z_j \right) \end{array} \right\}$$

Solution (2nd subcondition)

- Lagrangian:

$$L(\mathbf{w},\mathbf{z},\lambda) = E(z_{H+1},t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h\left(a_j\right) \right] + \lambda_{H+1}\left[z_{H+1} - h\left(a_{H+1}\right) \right]$$

- 2nd subcondition:

$$\forall j : \frac{\partial L}{\partial z_j} = 0 \quad \Rightarrow \quad \left\{ \begin{array}{l} \forall j : \quad \lambda_j - \lambda_{H+1} \dfrac{\partial h(a_{H+1})}{\partial z_j} = 0 \\ \dfrac{\partial E(z_{H+1},t)}{\partial z_{H+1}} + \lambda_{H+1} = 0 \end{array} \right\}$$

$$Backward\ dynamics : \left\{ \begin{array}{l} \forall j, \quad \lambda_j = \lambda_{H+1} \dfrac{\partial h(\mathbf{w}^{(2)},\mathbf{z})}{\partial z_j} \\ \lambda_{H+1} = -\dfrac{\partial E(z_{H+1},t)}{\partial z_{H+1}} \end{array} \right\}$$

Solution (3rd subcondition)

- Lagrangian:

$$L(\mathbf{w},\mathbf{z},\lambda) = E(z_{H+1},t) + \sum_{j=1}^{H} \lambda_j \left[z_j - h(a_j) \right] + \lambda_{H+1} \left[z_{H+1} - h(a_{H+1}) \right]$$

- 3rd subcondition:

$$\forall i,j: \frac{\partial L}{\partial w_{ij}} = 0 \quad \Rightarrow \quad \left\{ \begin{array}{l} \forall i,j: \quad -\lambda_j \dfrac{\partial h(\mathbf{w},\mathbf{x})}{\partial w_{ji}} = 0 \\ \forall j: \quad -\lambda_{H+1} \dfrac{\partial h(\mathbf{w}^{(2)},\mathbf{z})}{\partial w_j^{(2)}} = 0 \end{array} \right\}$$

$$Gradient\ descent: \left\{ \begin{array}{l} \forall i,j: \quad w_{ji} = w_{ji} - \eta \dfrac{\partial L}{\partial w_{ij}} \\ \forall j: \quad w_j^{(2)} = w_j^{(2)} - \eta \dfrac{\partial L}{\partial w_j^{(2)}} \end{array} \right\}$$

Back-Propagation Algorithm

1. Forward Pass (propagate)
2. Backward Pass (propagate)
3. Weight update

$$\left\{ \begin{array}{l} \forall j, \quad z_j = h\left(\sum_{i=1}^{D} w_{ji} x_i \right) \\ z_{H+1} = h\left(\sum_{j=1}^{H} w_j^{(2)} z_j \right) \end{array} \right\}$$

$$Backward\ dynamics: \left\{ \begin{array}{l} \forall j, \quad \lambda_j = \lambda_{H+1} \dfrac{\partial h(\mathbf{w}^{(2)}, \mathbf{z})}{\partial z_j} \\ \lambda_{H+1} = -\dfrac{\partial E(z_{H+1}, t)}{\partial z_{H+1}} \end{array} \right\}$$

$$Stochastic\ GD: \left\{ \begin{array}{l} \forall i, j: \quad w_{ji} = w_{ji} + \eta \lambda_j \dfrac{\partial h(\mathbf{w}, \mathbf{x})}{\partial w_{ji}} \\ \forall j: \quad w_j^{(2)} = w_j^{(2)} + \eta \lambda_{H+1} \dfrac{\partial h(\mathbf{w}^{(2)}, \mathbf{z})}{\partial w_j^{(2)}} \end{array} \right\}$$

Activation Function

- General Sigmoid functions: smooth monotonic such that $h(-\infty) = -1, \quad h(+\infty) = 1$
- Example: Logistic Sigmoid $h(a) = \left[1 + \exp(-a)\right]^{-1}$
- Consider hyperbolic tangent: $h(a) = \tanh(a)$

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

- Nice property: $h'(a) = 1 - h(a)^2$
- Assume activations on hidden layer are sigmoid (tanh), on output layer linear:

$$\frac{\partial h(\mathbf{w}^{(2)}, \mathbf{z})}{\partial w_j^{(2)}} = z_j \qquad \frac{\partial h(\mathbf{w}, \mathbf{x})}{\partial w_{ji}} = x_i$$

$$\frac{\partial h(\mathbf{w}^{(2)}, \mathbf{z})}{\partial z_j} = w_j^{(2)} h'\left(a_j\right) = w_j^{(2)}(1 - z_j^2)$$

Particular Error (Loss) Function

- Consider an obvious choice for a loss function:

$$E(z_{H+1}, t) = \frac{1}{2}(t - z_{H+1})^2$$

$$\frac{\partial E(z_{H+1}, t)}{\partial z_{H+1}} = -(t - z_{H+1})$$

- Backward dynamics simplified:

$$\lambda_{H+1} = (t - z_{H+1}) h'\left(a_{H+1}\right) = (t - z_{H+1})$$

Back-Propagation Algorithm

1. Forward Pass (propagate)
2. Backward Pass (propagate)
3. Weight update

$$\left\{\begin{array}{l} \forall j, \quad z_j = \tanh\left(\sum_{i=1}^{D} w_{ji} x_i\right) \\ z_{H+1} = \left(\sum_{j=1}^{H} w_j^{(2)} z_j\right) \end{array}\right\}$$

$$Backward\ dynamics: \left\{\begin{array}{l} \lambda_{H+1} = (t - z_{H+1}) \\ \forall j, \quad \lambda_j = \lambda_{H+1} w_j^{(2)} (1 - z_j^2) \end{array}\right\}$$

$$Stochastic\ GD: \left\{\begin{array}{l} \forall i, j: \quad w_{ji} = w_{ji} + \eta \lambda_j x_i \\ \forall j: \quad w_j^{(2)} = w_j^{(2)} + \eta \lambda_{H+1} z_j \end{array}\right\}$$

Shortcomings of NNs

1. Hard to interpret, black box, what are hidden layers doing?
2. Back-Prop can overfit
3. Multiple solution (weight space symmetries)
4. Many local minima (quality of solution depends on initialization, learning rate)
5. Convergence can be slow
6. Performance depends on heuristics, not "well controlled"

- But good results!
- Digits Demo: Le Net…http://yann.lecun.com

Improving NNs

• Since a single hidden layer neural network can approximate any function (universal approximator), it should be good for any application, right?

• NN with a single hidden layer is not an efficient approximator: construction requires an exponential number of units to be a universal approximator

• Research didn't focus on the best feature of NNs: depth. Depth reflects several steps of computation. NN can represent such functions efficiently (deep architectures/ learning)

• Intuition why deep neural networks work well for vision: many believe that to recognize an object, we need a sequence of computation steps. The idea is to extract progressively more abstract and specific units at each step:

1. Extract edges from image
2. Small parts (corners) should be computed from the edges
3. Compute combinations of small parts

• Speech & Object Recognition examples:

- http://www.cs.toronto.edu/~hinton/absps/DNN-2012-proof.pdf
- http://www.cs.utoronto.ca/~ilya/pubs/2012/imgnet.pdf