<table>
<thead>
<tr>
<th>(Unsupervised)</th>
<th>Feature Learning +</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Supervised)</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>(Self Taught)</td>
<td>Deep Learning</td>
</tr>
</tbody>
</table>

**Lecture 04**

Razvan C. Bunescu
School of Electrical Engineering and Computer Science

*bnescu@ohio.edu*
Self-Taught Learning

1. **Learn features:**
   1. Train sparse auto-encoder on unlabeled examples.
   2. Remove last layer and its connections.
   \[ \Rightarrow W^{(1)} \]

2. **Plug features into favorite ML supervised algorithm:**
   1. Train softmax classifier on labeled examples.
   \[ \Rightarrow W^{(2)} \]

3. **Fine-tune both \( W^{(1)} \) and \( W^{(2)} \):**
   1. Train whole network (first layer + softmax) on labeled examples.
   2. Start parameter learning from **pre-trained** \( W^{(1)}, W^{(2)} \)
1. Learning Features

- Input: $x_1, x_2, x_3, x_4, x_5, x_6$
- Features: $\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6$
- Output: $a_1, a_2, a_3$

Remove output layer.
2. Plug Features into Softmax Regressor

\[ P(y = 0 \mid x) \]
3. Fine-Tuning

\[ P(y = 0 \mid x) \]
Shallow vs. Deep Networks

• A 1-hidden layer network is a fairly shallow network.
  – Effective for MNIST, but limited by simplicity of features.

• A deep network is a $k$-layer network, $k > 1$.
  – Computes more complex features of the input, as $k$ gets larger.
  – Each hidden layer computes a non-linear transformation of the previous layer.

A deep network has significantly greater representational power than a shallow one.
Deep vs. Shallow Networks

- A function is **highly varying** when a piecewise (linear) approximation would require a large number of pieces.

- **Depth** of an architecture refers to the number of levels of composition of non-linear operations in the function computed by the architecture.

- **Deep architectures** can **compactly** represent **highly-varying functions**:
  - The expression of a function is **compact** when it has few computational elements.
  - Same highly-varying functions would require very large shallow networks.
Graphs of Computations

• A function can be expressed by the composition of **computational elements** from a given **set**:
  – logic operators.
  – logistic operators.
  – multiplication and additions.

• The function is defined by a **graph of computations**:
  – A directed acyclic graph, with one node per computational element.
  – Depth of architecture = depth of the graph = longest path from an input node to an output node.
Functions as Graphs of Computations

[Bengio, FTML’09]
Polynomials as Graphs of Computations

\[(x_1x_2)(x_2x_3) + (x_1x_2)(x_3x_4) + (x_2x_3)^2 + (x_2x_3)(x_3x_4)\]

\[(x_1x_2) + (x_2x_3) (x_2x_3) + (x_3x_4)\]

\[x_1x_2 \quad x_2x_3 \quad x_3x_4\]

\[x_1 \quad x_2 \quad x_3 \quad x_4\]
Polynomials as Graphs of Computations

- **Deep Architecture** (previous slide, nodes with 2 inputs):
  - 3 layers + 1 input layer:
    - 12 links.
    - 6 nodes.

- **Shallow Architecture** (nodes with 2 or more inputs):
  - 2 layers + 1 input layer:
    - 20 links.
    - 5 nodes.

=> Deep Architecture leads to a more *compact* representation.
Graphs of Computations

- If we include affine operations and their possible composition with sigmoids in the set of computational elements, linear regression and logistic regression have depth 1, i.e., have a single level.

- When we put a fixed kernel computation $K(u, v)$ in the set of allowed operations, along with affine operations, kernel machines (Schölkopf, Burges, & Smola, 1999a) with a fixed kernel can be considered to have two levels. The first level has one element computing $K(x, x_i)$ for each prototype $x_i$ (a selected representative training example) and matches the input vector $x$ with the prototypes $x_i$. The second level performs an affine combination $b + \sum_i \alpha_i K(x, x_i)$ to associate the matching prototypes $x_i$ with the expected response.

- When we put artificial neurons (affine transformation followed by a non-linearity) in our set of elements, we obtain ordinary multi-layer neural networks (Rumelhart et al., 1986b). With the most common choice of one hidden layer, they also have depth two (the hidden layer and the output layer).

- Boosting (Freund & Schapire, 1996) usually adds one level to its base learners: that level computes a vote or linear combination of the outputs of the base learners.

- Stacking (Wolpert, 1992) is another meta-learning algorithm that adds one level.

- Based on current knowledge of brain anatomy (Serre et al., 2007), it appears that the cortex can be seen as a deep architecture, with 5 to 10 levels just for the visual system.
Computational Complexity

• When a function can be compactly represented by a deep architecture, it might need a very large architecture to be represented by an insufficiently deep one.

A two-layer circuit of logic gates can represent any Boolean function (Mendelson, 1997). Any Boolean function can be written as a sum of products (disjunctive normal form: AND gates on the first layer with optional negation of inputs, and OR gate on the second layer) or a product of sums (conjunctive normal form: OR gates on the first layer with optional negation of inputs, and AND gate on the second layer). To understand the limitations of shallow architectures, the first result to consider is that with depth-two logical circuits, most Boolean functions require an exponential (with respect to input size) number of logic gates (Wegener, 1987) to be represented.

More interestingly, there are functions computable with a polynomial-size logic gates circuit of depth $k$ that require exponential size when restricted to depth $k - 1$ (Håstad, 1986). The proof of this theorem relies on earlier results (Yao, 1985) showing that $d$-bit parity circuits of depth 2 have exponential size. The $d$-bit parity function is defined as usual:

$$\text{parity} : (b_1, \ldots, b_d) \in \{0, 1\}^d \mapsto \begin{cases} 1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\ 0 & \text{otherwise}. \end{cases}$$
One might wonder whether these computational complexity results for Boolean circuits are relevant to machine learning. See Orponen (1994) for an early survey of theoretical results in computational complexity relevant to learning algorithms. Interestingly, many of the results for Boolean circuits can be generalized to architectures whose computational elements are *linear threshold units* (also known as artificial neurons (McCulloch & Pitts, 1943)), which compute

\[ f(x) = 1_{w'x + b \geq 0} \]

(1)

with parameters \( w \) and \( b \). The *fan-in* of a circuit is the maximum number of inputs of a particular element. Circuits are often organized in layers, like multi-layer neural networks, where elements in a layer only take their input from elements in the previous layer(s), and the first layer is the neural network input. The *size* of a circuit is the number of its computational elements (excluding input elements, which do not perform any computation).

Of particular interest is the following theorem, which applies to *monotone weighted threshold circuits* (i.e. multi-layer neural networks with linear threshold units and positive weights) when trying to represent a function compactly representable with a depth \( k \) circuit:

**Theorem 2.1.** A monotone weighted threshold circuit of depth \( k - 1 \) computing a function \( f_k \in \mathcal{F}_{k,N} \) has size at least \( 2^{cN} \) for some constant \( c > 0 \) and \( N > N_0 \) (Håstad & Goldmann, 1991).