Deep Learning: Self-Taught Learning and Deep vs. Shallow Architectures

Lecture 04

Razvan C. Bunescu
School of Electrical Engineering and Computer Science
bunescu@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 | Features | Output
---|---|---
$x_1$ | $\hat{x}_1$ | 
$x_2$ | $\hat{x}_2$ | 
$x_3$ | $\hat{x}_3$ | 
$x_4$ | $\hat{x}_4$ | 
$x_5$ | $\hat{x}_5$ | 
$x_6$ | $\hat{x}_6$ | 
+1 | | 

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.

Conjecture

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

Conjecture

A deep network has significantly greater representational power than a shallow one.

Figure 1: Binary classification using a shallow model with 20 hidden units (solid line) and a deep model with two layers of 10 units each (dashed line). The right panel shows a close-up of the left panel. Filled markers indicate errors made by the shallow model.
Deep vs. Shallow Architectures

• 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.

• **Conjecture:** 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 \textit{computational elements} from a given set:
  – logic operators.
  – logistic operators.
  – multiplication and additions.

• The function is defined by a \textit{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

\[(x_1 x_2)(x_2 x_3) + (x_1 x_2)(x_3 x_4) + (x_2 x_3)^2 + (x_2 x_3)(x_3 x_4)\]

\[(x_1 x_2) + (x_2 x_3)\]

\[(x_2 x_3) + (x_3 x_4)\]

\[x_1 \times x_2\]

\[x_2 \times x_3\]

\[x_3 \times x_4\]

\[x_1 \quad x_2 \quad x_3 \quad x_4\]
Sum-Product Networks (SPNs)

- Rooted, weighted DAG.
- **Nodes**: Sum, Product, (Input) Indicators.
- **Weights** on edges from sums to children.

---

**Example Table:**

<table>
<thead>
<tr>
<th>X</th>
<th>(X_1 = 1, X_2 = 0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(X_1)</td>
<td>1</td>
</tr>
<tr>
<td>(\overline{X_1})</td>
<td>0</td>
</tr>
<tr>
<td>(X_2)</td>
<td>0</td>
</tr>
<tr>
<td>(\overline{X_2})</td>
<td>1</td>
</tr>
</tbody>
</table>

**Example Diagram:**

```
X: X_1 = 1, X_2 = 0

\(\overline{X_1}\) 0.6 \(\overline{X_1}\)
\(X_1\) 0.4 \(X_1\)
\(\overline{X_2}\) 1.0 \(\overline{X_2}\)
\(X_2\) 0.9 \(X_2\)

0.7 0.3 0.7 0.8
0.6 0.4 0.9 0.1
0.2 0.8 0.7 0.3
```

[12] Lecture 4

[12] Poon & Domingos, UAI’11
ML Models as 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.
Deep vs. Shallow Architectures

- 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:

\[
\text{parity} : (b_1, \ldots, b_d) \in \{0, 1\}^d \mapsto \begin{cases} 
1 & \text{if } \sum_{i=1}^{d} b_i \text{ is even} \\
0 & \text{otherwise.}
\end{cases}
\]
Deep vs. Shallow Architectures

- Many of the results for Boolean circuits can be generalized to architectures whose computational elements are *linear threshold* units i.e. Mc-Cullogh & Pitts neurons:
  \[ f(x) = \mathbf{1}[\mathbf{w}^T x + b \geq 0] \]

- *Monotone weighted threshold circuits* = multi-layer neural networks with linear threshold units and positive weights.

*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).

The class of functions \( \mathcal{F}_{k,N} \) is defined as follows. It contains functions with \( N^{2k-2} \) inputs, defined by a depth \( k \) circuit that is a tree. At the leaves of the tree there are unnegated input variables, and the function value is at the root. The \( i \)-th level from the bottom consists of \( \text{AND} \) gates when \( i \) is even and \( \text{OR} \) gates when \( i \) is odd. The fan-in at the top and bottom level is \( N \) and at all other levels it is \( N^2 \).
Deep vs. Shallow Architectures

• Deep architectures were shown to be more compact for:
  – *Boolean circuits* [Hastad, 1986].
  – *Monotone weighted threshold circuits* [Hastad and Goldman, 1993].

• Same holds for *networks with continuous-valued activations* [Maass, 1992].

• Many modern neural networks use rectified linear units:
  1. *ReLU networks* are universal approximators [Leshno et al., 1993].
  2. Are deep ReLU networks more compact than shallow ones?
     • **YES!** [Montufar et al., NIPS’14]
ReLU and Generalizations

• It has become more common to use piecewise linear activation functions for hidden units:
  – **ReLU**: the rectifier activation \( g(a) = \max\{0, a\} \).
  – **Absolute value ReLU**: \( g(a) = |a| \).
  – **Maxout**: \( g(a_1, \ldots, a_k) = \max\{a_1, \ldots, a_k\} \).
    • needs \( k \) weight vectors instead of 1.
  – **Leaky ReLU**: \( g(a) = \max\{0, a\} + \alpha \min(0, a) \).

⇒ the network computes a *piecewise linear function* (up to the output activation function).
ReLU vs. Sigmoid and Tanh

- Sigmoid and Tanh saturate for values not close to 0:
  - “kill” gradients, bad behavior for gradient-based learning.
- ReLU does not saturate for values > 0:
  - greatly accelerates learning, fast implementation.
  - fragile during training and can “die”, due to 0 gradient:
    - initialize all $b$’s to a small, positive value, e.g. 0.1.
ReLU vs. Softplus

- Softplus $g(a) = \ln(1+e^a)$ is a smooth version of the rectifier.
  - Saturates less than ReLU, yet ReLU still does better [Glorot, 2011].
ReLU and Generalizations

- Leaky ReLU attempts to fix the “dying” ReLU problem.
- Maxout subsumes (leaky) ReLU, but needs more params.
Maxout Networks

Maxout units can learn the activation function.

Figure 1. Graphical depiction of how the maxout activation function can implement the rectified linear, absolute value rectifier, and approximate the quadratic activation function. This diagram is 2D and only shows how maxout behaves with a 1D input, but in multiple dimensions a maxout unit can approximate arbitrary convex functions.
Number of Linear Regions of Shallow vs. Deep Networks

**Theorem**

A deep network has significantly greater representational power than a shallow one.

Figure 1: Binary classification using a shallow model with 20 hidden units (solid line) and a deep model with two layers of 10 units each (dashed line). The right panel shows a close-up of the left panel. Filled markers indicate errors made by the shallow model.
Folding Example

\[ x_1 \quad -1 \quad \quad -1 \quad +1 \quad x_2 \]

\[ \begin{array}{c}
0 \\
1
\end{array} \quad \begin{array}{c}
0 \\
1
\end{array} \]

Lecture 4
Folding Example

\[ x_1 \quad -1 \quad x_2 \quad -1 \quad +1 \]
Folding Example

\[ x_1 - 1 - 1 +1 \]

\[ x_2 -1 +1 \]

\[ 0 1 \]

\[ 2 0 \]
Folding Example

Lecture 4
Space Foldings

• Each hidden layer of a deep neural network can be associated with a folding operator.
Folding Example

Lecture 4

\[
\begin{align*}
x_1 & \rightarrow -1 \rightarrow +1 \\
& \downarrow \quad \quad \quad \quad \downarrow \\
& +1 \quad -1 +1 \\
\downarrow \quad \quad \quad \quad \downarrow \\
x_2 & \rightarrow -1 +1 \\
& \downarrow \quad \quad \quad \downarrow \\
& +1 -2 +1 \\
& \downarrow \quad \quad \downarrow \\
& -2 \\
\end{align*}
\]
Space foldings

- Each hidden layer of a deep neural network can be associated with a folding operator.
Folding Example

Lecture 4
Folding Example

Lecture 4
Folding Example

Lecture 4
Space Foldings

• Each hidden layer of a deep neural network can be associated with a folding operator:
  – Each hidden layer folds the space of activations of the previous layer.
  – In turn, a deep neural network effectively folds its input-space recursively, starting with the first layer.

• Any function computed on the final folded space will apply to all the collapsed subsets identified by the map corresponding to the succession of foldings.

• This means that in a deep model any partitioning of the last layer’s image-space is replicated in all input-space regions which are identified by the succession of foldings.

Montufar et al., NIPS’14
Space Foldings

Space foldings are not restricted to foldings along coordinate axes and they do not have to preserve lengths:

- The space is folded depending on the orientations and shifts encoded in:
  - The input weights $W$ and biases $b$.
  - The nonlinear activation function used at each hidden layer.

- The sizes and orientations of identified input-space regions may differ from each other.

- For activation functions which are not piece-wise linear, the folding operations may be even more complex.

[Montufar et al., NIPS’14]
Space Foldings

[Montufar et al., NIPS’14]
Space Foldings

• Space folding of 2-D space in a non-trivial way:
  – The folding can potentially identify symmetries in the boundary that it needs to learn.

[Montufar et al., NIPS’14]
Deep vs. Shallow Rectifier Networks

A *linear region* of a piecewise linear function $F: \mathbb{R}^d \rightarrow \mathbb{R}^m$ is a maximal connected subset of the input-space $\mathbb{R}^d$, on which $F$ is linear.

- The number of linear regions carved out by a *deep rectifier network* with $d$ inputs, depth $l$, and $n$ units per hidden layer, is:

$$O \left( \binom{n}{d}^{d(l-1)} n^d \right)$$

- In the case of *maxout networks* with $k$ filters per unit, the number of linear regions is:

$$O \left( k^{(l-1)+d} \right)$$

[Montufar et al., NIPS’14]
Why are Deep Architectures Good for AI?

• There is no guarantee that the kinds of functions we want to learn share this “folding” property.
• Choosing a deep model encodes a very general belief that:
  – The function we want to learn should involve composition of several simpler functions, OR
  – The learning problem consists of discovering a set of underlying factors of variation that can in turn be described in terms of other, simpler underlying factors of variation, OR
  – The function we want to learn is a computer program consisting of multiple steps, where each step uses of the previous step’s output.
• Empirically, greater depth does seem to result in better generalization for a wide variety of tasks.
Figure 6.6: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses. Data from Goodfellow et al. (2014d). The test set accuracy consistently increases with increasing depth. See figure 6.7 for a control experiment demonstrating that other increases to the model size do not yield the same effect.
Figure 6.7: Deeper models tend to perform better. This is not merely because the model is larger. This experiment from Goodfellow et al. (2014d) shows that increasing the number of parameters in layers of convolutional networks without increasing their depth is not nearly as effective at increasing test set performance. The legend indicates the depth of network used to make each curve and whether the curve represents variation in the size of
Why do Cheap & Deep Architectures Work for AI?

[Lin & Tegmark, 2016]

- **Paradox**: How can neural networks approximate functions well in practice, when the set of possible functions is exponentially larger than the set of practically possible networks?
  - **Example**: classify megapixel greyscale images into two categories, e.g., cats or dogs.
  - If each pixel can take one of 256 values, then there are $256^{1000000}$ possible images, and for each one, we wish to compute the probability that it depicts a cat.
  - This means that an arbitrary function is defined by a list of $256^{1000000}$ probabilities, i.e., way more numbers than there are atoms in our universe (about $10^{78}$).

- **Paradox**: How can neural networks approximate functions well in practice, when the set of possible functions is exponentially larger than the set of practically possible networks?

- **Conjecture**: The data sets and functions we care about form a minuscule minority, and it is plausible that they can also be efficiently implemented by neural networks reflecting their generative process.

1. **Cheap**: The exceptional simplicity of physics-based functions hinges on properties such as **symmetry**, **locality**, **compositionality** and **polynomial log-probability**.
   - These properties translate into exceptionally simple neural networks approximating both natural phenomena such as images and abstract representations thereof such as drawings.

2. **Deep**: The statistical process generating the data is of a certain **hierarchical** form prevalent in physics and machine learning:
   - Therefore, a deep neural network can be more efficient than a shallow one.
Why do Cheap Architectures Work for AI?

[Lin & Tegmark, 2016]

- **Low polynomial order**: For reasons that are still not fully understood, our universe can be accurately described by polynomial Hamiltonians of low order $d$.
  - **Standard model**: At a fundamental level, the Hamiltonian of the standard model of particle physics has $d = 4$.
  - **Central Limit Theorem**: many probability distributions in machine-learning and statistics can be accurately approximated by multivariate Gaussians $\Rightarrow$ Hamiltonian $H = -\ln p$ has $d = 2$.

- **Translation and rotation invariance**.
- **Locality**.
- **Symmetry**.
Why do Cheap Architectures Work for AI? [Lin & Tegmark, 2016]

- **Low polynomial order**: For reasons that are still not fully understood, our universe can be accurately described by polynomial Hamiltonians of low order $d$.

- Neural networks can efficiently approximate multiplication!
Polynomials as Simple NNs

![Diagram](Lin & Tegmark, 2016]

Fig. 2: Multiplication can be efficiently implemented by simple neural nets, becoming arbitrarily accurate as $\lambda \rightarrow 0$ (left) and $\beta \rightarrow \infty$ (right). Squares apply the function $\sigma$, circles perform summation, and lines multiply by the constants labeling them. The “1” input implements the bias term. The left gate requires $\sigma''(0) \neq 0$, which can always be arranged by biasing the input to $\sigma$. The right gate requires the sigmoidal behavior $\sigma(x) \rightarrow 0$ and $\sigma(x) \rightarrow 1$ as $x \rightarrow -\infty$ and $x \rightarrow \infty$. 
Why do Deep Architectures Work for AI?

• **Hierarchical Structure**: One of the most striking features of the physical world is its hierarchical structure.
  - **Spatially**, it is an object hierarchy: elementary particles form atoms which in turn form molecules, cells, organisms, planets, solar systems, galaxies, etc.
  - **Causally**, complex structures are frequently created through a distinct sequence of simpler steps.
Why do Cheap & Deep Architectures Work for AI?  

[Lin & Tegmark, 2016]

- **Cheap & Deep:**
  - **Paradox:** The number of parameters required to describe an arbitrary function of the input data $y$ is beyond astronomical.
  - **Solution:** The generative process can be specified by a more modest number of parameters, because each of its steps can.
    - For a megapixel image of a galaxy, its entire prob. distribution is defined by the standard model of particle physics with its 32 parameters, which together specify the process transforming primordial hydrogen gas into galaxies.
    - Giving the simple low-information content instruction “draw a cute kitten” to a random sample of artists will produce a wide variety of images $y$ with a complicated probability distribution over colors, postures, etc. But the pre-stored information about cat probabilities in these artists' brains is modest in size.