CS 615 – Deep Learning

An Introduction to Deep Learning

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- What/Why Deep Learning
- Issues with Deep Learning
- Multi-Layer Perceptrons (MLPs)

Intro to Deep Learning

Why Deep Learning?

- The success of a traditional machine learning algorithm depends heavily on “pre-processing”.
- That is, taking the raw data and extracting useful features from it.
- However, handcrafting features is time-consuming, largely chosen via experimentation, and varies for each task/domain
- Deep learning attempts to take the “human out of the loop”.
- That is, it looks to learn how to extract useful information from the raw data at early layers, to be used for later layers.

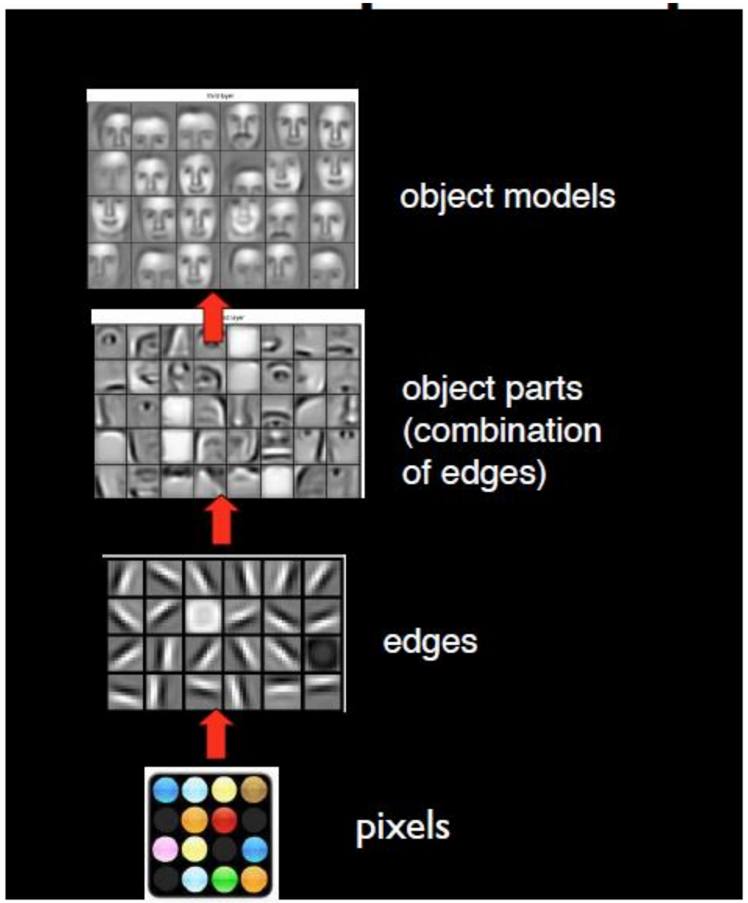
Deep Learning Example

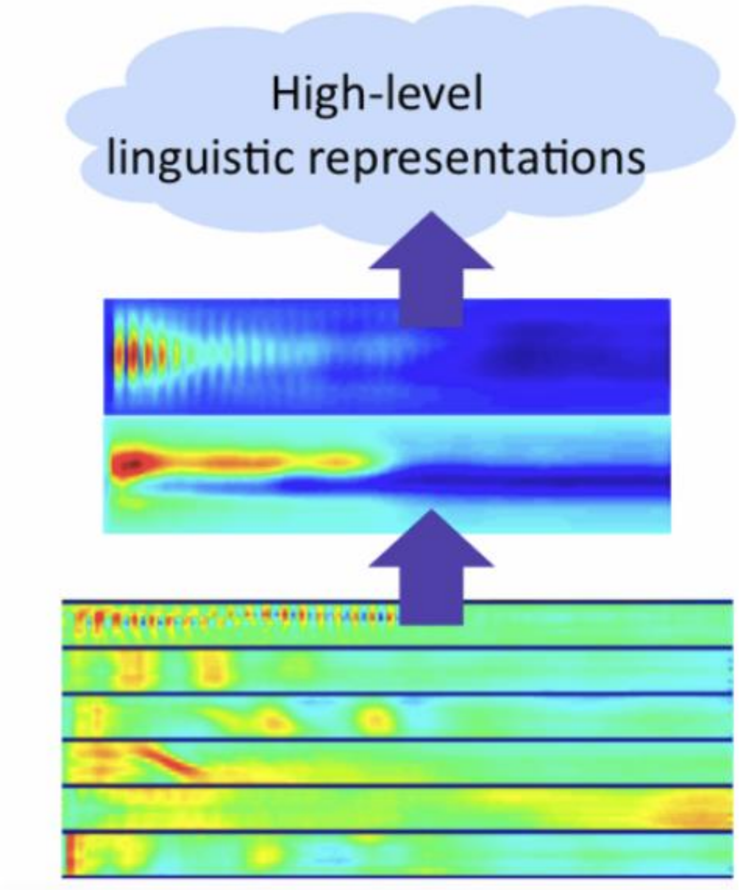
- As an example, let's imagine the tasks of face recognition as a multi-class classification problem.
- Our raw data is the pixel data.
- We could try to use that, but there could be issues:
    - It may be noisy.
    - It's not invariant to changes in location, illumination ,scale, etc..
- Therefore to have success, we might want to first extract a more useful representation.
    - Histogram
    - Histogram of Oriented Gradients
    - Bag of SIFT features
    - Etc…
- Or let's let a deep learning do the "work for us"!

Deep Learning Example

- Although just a “thought experiment”, imagine the following:
  - At the input, we just have pixels.
  - At the first hidden layer, we determine ways to combine pixels that will be useful later on.
    - Perhaps we learn the idea of edges.
  - At the next layer, we determine ways to combine output from the previous layer, again in such a way to be useful later.
    - Perhaps here we learn the idea of parts (combos of edges)
  - And so on and so on until we get to our output layer.

Deep Learning Example





Issues with Deep Learning

- It's not all rainbows and unicorns though ☹

Image from Church Unlimited

Issues with Deep Learning

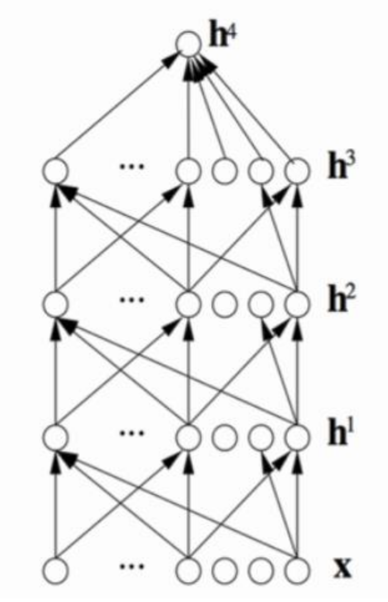
- Now we have even more “design decisions” to make!
- How many layers?
- How many nodes in each layer?
- What activation function to use at each layer?
- And of course
  - What is our objective function?
  - What is our learning rate?
- Not to mentioned training time and potential overfitting….

Multi Layer ANN

- The most natural dive into deep networks is to generalize our artificial neural network to be able to have an arbitrary number of hidden layers.
- This is where the chain rule, and the fact that some of the terms from the previous gradient are used in the next gradient, becomes crucial.
- Recall:

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \theta_{jk} \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \right) x_i$$



Multiple Hidden Layers

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot h_j = \delta_{jk} \cdot h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \theta_{jk} \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \right) x_i = \sum_{k=1}^{K} \left(\delta_{jk} \cdot \theta_{jk} \cdot \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \right) x_i = \delta_{ij} x_i$$

- So to “generalize” the gradient rules we can say:
    1. Find the gradient for the parameters leading to the output layer, storing δ_{jk} for use in next layer.
    2. Compute the gradient for the next parameter set using
        1. The δ from the previous layer.
        2. The gradient of this layer’s activation function.
        3. The inputs from the next layer.
    3. Store as this layer’s δ the product of all but the last of these terms.
    4. Repeat 2-3 until we reach the input layer.

Multiple Hidden Layers

- Let $\frac{\partial J}{\partial \theta^{(m)}}$ be the gradient rule for layer m
- Let there be M total layers (including the output and input layers).
- We can recursively/iteratively compute $\frac{\partial J}{\partial \theta^{(m)}}$ as
  - For $m = M, \dots, 2$
    - If $m = M$ //output layer
      - $\delta = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o}$
    - Otherwise
      - $\delta = \left(\delta \theta^{(m+1)^T}\right) \circ \frac{\partial g(net_m)}{\partial net_m}$
    - $\frac{\partial J}{\partial \theta^{(m)}} = \frac{1}{N} g(net_{m-1})^T \delta$