CSE 40537 / 60537: Biometrics

Face Recognition 10
Deep Learning for Face Recognition
Limitations of Hand-tuned Features

- The representations we’ve discussed are not strongly invariant
  - Need to go beyond V1

Solution: Multi-layer Neural Networks
Feature Learning

- Perhaps get better performance?
- **Deep models**: hierarchy of feature extractors
- All the way from pixels to the classifier
- One layer extracts features from output of previous layer

Train all layers jointly

---

Adapted from tutorial by R. Fergus
The diagram illustrates the classification of different machine learning models into deep and shallow categories. The supervised learning models include Recurrent Neural Net, Neural Net, and Convolutional Neural Net. The unsupervised learning models include Deep (sparse/denoising), Deep Belief Net, BayesNP, and others. The slide is credited to M. Ranzato.
Multi-stage Hubel-Wiesel Architecture

Hubel & Wiesel 1962
• Simple cells detect local features
• Complex cells “pool” the outputs of simple cells within a retinotopic neighborhood

Fukushima 1971-1982
• Cognitron / Neocognitron

Poggio 2002 - 2006
• HMAX

LeCun 1988 - present
• Convolutional Networks
Perceptrons

“Integrate and Fire” Model
Artificial Neural Networks

Artificial “neurons” are connected together to form a network

- Contains sets of adaptive weights
  - These are learned during training
- Capable of approximating non-linear functions of their input
Non-linearity

- Single layer perceptrons cannot model non-linear functions
- Adding a hidden layer with a non-linear activation function addresses this
Gradients and Training

• We need a strategy to set the weights with respect to observed error on the training set

1. Initialize $w(0)$
2. For $t = 0, 1, 2, \ldots$ [to termination]
3. $w(t + 1) = w(t) - \alpha \nabla E_{in}(w(t))$
4. Return final $w$

$\alpha =$ learning rate

Image credit: Yaser S. Abu-Mostafa, *Learning From Data*
Gradients and Training

Stochastic Gradient Descent: some randomness helps

- $\nabla E_{in}$ is based on all training examples $(x_n, y_n)$
- Pick one $(x_n, y_n)$ at a time. Apply GD to $e(f(x_n), y_n)$

$$- \nabla E_{in} = \frac{1}{N} \sum_{n=1}^{N} - \nabla e(f(x_n), y_n)$$

Image credit: Yaser S. Abu-Mostafa, Learning From Data
Backpropagation

What’s an efficient way to calculate the gradients?

The gradient of a set of nested functions is the product of the individual derivatives:

\[
\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \frac{\partial f_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial x}
\]

Gradients are matrices of all first order partial derivatives of \(f_n\) (Jacobian)

\[
\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \frac{\partial f_4}{\partial f_3} \cdot \left( \frac{\partial f_3}{\partial f_2} \cdot \left( \frac{\partial f_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial x} \right) \right)
\]

Bad: For vectors of dimensionality \(D\), need to propagate \(D \times D\) Jacobian at each step
Backpropagation

\[
\frac{\partial f_4(f_3(f_2(f_1(x))))}{\partial x} = \left( \left( \frac{\partial f_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial f_2} \right) \cdot \frac{\partial f_2}{\partial f_1} \right) \cdot \frac{\partial f_1}{\partial x}
\]

Good: accumulate just a \(D\)-dimensional vector at each step by starting with scalar output.

But… typical implementations store the entire training trajectory, \(w_1 \ldots w_t\) in memory.

- Resource intensive for a single feed-forward deep network optimizing weights.
What is deep learning?

The simple answer:

**Just multilayer artificial neural networks!**
Back to those filters...

What is a Convolution? → Image filtering
Convolution Operator

That was the result of applying this mask:

\[ w(x, y) \ast f(x, y) = \sum_{i=\lceil -m/2 \rceil}^{\lceil m/2 \rceil} \sum_{j=\lceil -n/2 \rceil}^{\lceil n/2 \rceil} w(i, j) f(x - i, y - j) \]  

\textbf{OR} the dot product