Deep Learning

- CSCE 636 Neural Networks
- Instructor: Yoonsuck Choe

What Is Deep Learning?

- Learning higher level abstractions/representations from data.
- Motivation: how the brain represents sensory information in a hierarchical manner.

The Rise of Deep Learning

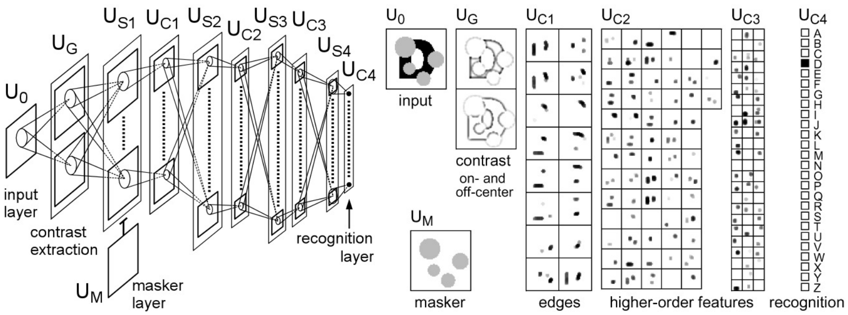
Made popular in recent years

- Geoffrey Hinton et al. (2006).
- Andrew Ng & Jeff Dean (Google Brain team, 2012).
- Schmidhuber et al.'s deep neural networks (won many competitions and in some cases showed super human performance; 2011–).

Long History (in Hind Sight)

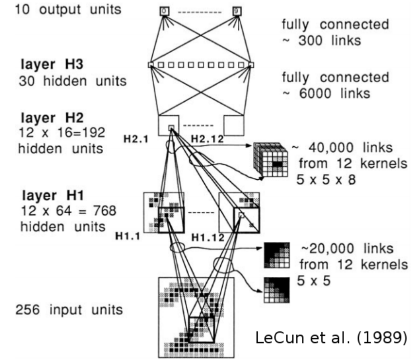
- Fukushima's Neocognitron (1980).
- LeCun et al.'s Convolutional neural networks (1989).
- Schmidhuber's work on stacked recurrent neural networks (1993). Vanishing gradient problem.

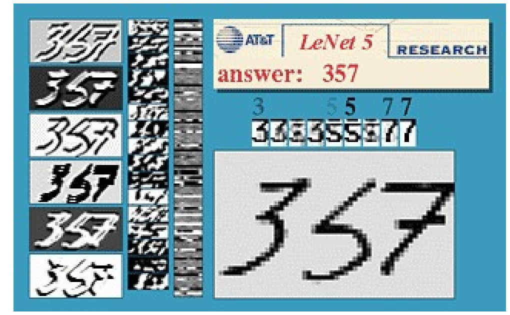
Fukushima's Neocognitron



- Appeared in journal *Biological Cybernetics* (1980).
- Multiple layers with local receptive fields.
- S cells (trainable) and C cells (fixed weight).
- Deformation-resistent recognition.

LeCun's Colvolutional Neural Nets





- Convolution kernel (weight sharing) + Subsampling
- Fully connected layers near the end.

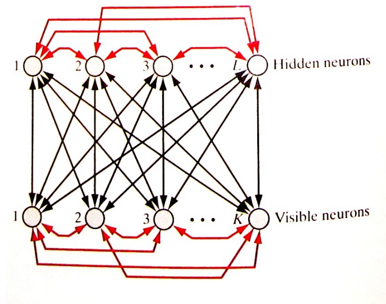
Current Trends

- Deep belief networks (based on Boltzmann machine)
- Deep neural networks
- Convolutional neural networks
- Deep Q-learning Network (extensions to reinforcement learning)

Boltzmann Machine to Deep Belief Nets

- Haykin Chapter 11: Stochastic Methods rooted in statistical mechanics.

Boltzmann Machine



- Stochastic binary machine: +1 or -1.
- Fully connected symmetric connections: $w_{ij} = w_{ji}$.
- Visible vs. hidden neurons, clamped vs. free-running.
- Goal: Learn weights to model prob. dist of visible units.
- Unsupervised. Pattern completion.

Boltzmann Machine: Energy

- Network state: $\mathbf{x}$ from random variable $\mathbf{X}$.
- $w_{ij} = w_{ji}$ and $w_{ii} = 0$.
- Energy (in analogy to thermodynamics):

$$E(\mathbf{x}) = -\frac{1}{2}\sum_{i}\sum_{j,i\neq j} w_{ji}x_ix_j$$

Boltzmann Machine: Prob. of a State $\mathbf{x}$

- Probability of a state $\mathbf{x}$ given $E(\mathbf{x})$ follows the *Gibbs distribution*:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z}\exp\left(-\frac{E(\mathbf{x})}{T}\right),$$

  - Z: *partition function* (normalization factor – hard to compute)

$$Z = \sum_{\forall \mathbf{x}} \exp(-E(\mathbf{x})/T)$$

  - T: temperature parameter.
  - Low energy states are exponentially more probable.
- With the above, we can calculate

$$P(X_j = x | \{X_i = x_i\}_{i=1,i\neq j}^{K})$$

  - This can be done without knowing Z.

Boltzmann Machine: $P(X_j = x|\text{the rest})$

- $A : X_j = x$. $B : \{X_i = x_i\}_{i=1,i\neq j}^{K}$ (the rest).

$$\begin{aligned}
P(X_j = x|\text{the rest}) &= \frac{P(A,B)}{P(B)} \\
&= \frac{P(A,B)}{\sum_A P(A,B)} = \frac{P(A,B)}{P(A,B) + P(\neg A, B)} \\
&= \frac{1}{1 + \exp\left(-\frac{x}{T}\sum_{i,i\neq j} w_{ji}x_i\right)} \\
&= \text{sigmoid}\left(\frac{x}{T}\sum_{i,i\neq j} w_{ji}x_i\right)
\end{aligned}$$

- Can compute equilibrium state based on the above.

Boltzmann Machine: Gibbs Sampling

- Initialize $\mathbf{x}^{(0)}$ to a random vector.
- For $j = 1, 2, ..., n$ (generate n samples $\mathbf{x} \sim P(\mathbf{X})$)
  - $x_1^{(j+1)}$ from $p(x_1|x_2^{(j)}, x_3^{(j)}, ..., x_K^{(j)})$
  - $x_2^{(j+1)}$ from $p(x_2|x_1^{(j+1)}, x_3^{(j)}, ..., x_K^{(j)})$
  - $x_3^{(j+1)}$ from $p(x_3|x_1^{(j+1)}, x_2^{(j+1)}, x_4^{(j)} ..., x_K^{(j)})$
  - ...
  - $x_K^{(j+1)}$ from $p(x_K|x_1^{(j+1)}, x_2^{(j+1)}, x_3^{(j+1)} ..., x_{K-1}^{(j+1)})$

  $\rightarrow$ One new sample $\mathbf{x}^{(j+1)} \sim P(\mathbf{X})$.
- Simulated annealing used (high T to low T) for faster conv.

Boltzmann Learning Rule (1)

- Probability of activity pattern being *one of* the training patterns (visible unit: subvector $\mathbf{x}_\alpha$; hidden unit: subvector $\mathbf{y}_\beta$), given the weight vector $\mathbf{w}$.

$$P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$$

- Log-likelihood of the visible units being *any one of* the trainning patterns (assuming they are mutually independent) $\mathcal{T}$:

$$\begin{aligned} L(\mathbf{w}) &= \log \prod_{\mathbf{x}_\alpha \in \mathcal{T}} P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \\ &= \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) \end{aligned}$$

- We want to learn $\mathbf{w}$ that **maximizes** $L(\mathbf{w})$.

Boltzmann Learning Rule (2)

- Want to calculate $P(\mathbf{X}_\alpha = \mathbf{x}_\alpha)$: use energy function.

$$\begin{aligned} P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) &= \frac{1}{Z} \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \\ \log P(\mathbf{X}_\alpha = \mathbf{x}_\alpha) &= \log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log Z \\ &= \log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \\ &\quad - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \end{aligned}$$

- Note: $Z = \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right)$

Boltzmann Learning Rule (3)

- Finally, we get:

$$L(\mathbf{w}) = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \left(\log \sum_{\mathbf{x}_\beta} \exp\left(-\frac{E(\mathbf{x})}{T}\right) - \log \sum_{\mathbf{x}} \exp\left(-\frac{E(\mathbf{x})}{T}\right) \right)$$

- Note that $\mathbf{w}$ is involved in:

$$E(\mathbf{x}) = -\frac{1}{2} \sum_i \sum_{j, i \neq j} w_{ji} x_i x_j$$

- Differentiating $L(\mathbf{w})$ wrt w_{ji}, we get:

$$\begin{aligned} \frac{\partial L(\mathbf{w})}{\partial w_{ji}} &= \frac{1}{T} \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \left(\sum_{\mathbf{x}_\beta} P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i \right. \\ &\quad \left. - \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) x_j x_i \right) \end{aligned}$$

Boltzmann Learning Rule (4)

- Setting:

$$\rho_{ji}^{+} = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}_\beta} P(\mathbf{X}_\beta = \mathbf{x}_\beta | \mathbf{X}_\alpha = \mathbf{x}_\alpha) x_j x_i$$

$$\rho_{ji}^{-} = \sum_{\mathbf{x}_\alpha \in \mathcal{T}} \sum_{\mathbf{x}} P(\mathbf{X} = \mathbf{x}) x_j x_i$$

- We get:

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \frac{1}{T} \left(\rho_{ji}^{+} - \rho_{ji}^{-} \right)$$

- Attempting to maximize $L(\mathbf{w})$, we get:

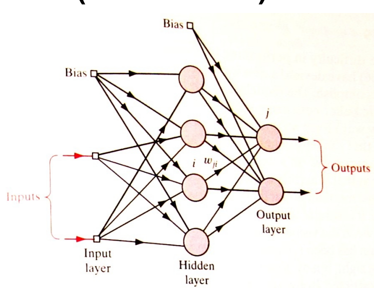
$$\Delta w_{ji} = \epsilon \frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \eta \left(\rho_{ji}^{+} - \rho_{ji}^{-} \right)$$

where $\eta = \frac{\epsilon}{T}$. This is *gradient ascent*.

Boltzmann Machine Summary

- Theoretically elegant.
- Very slow in practice (especially the unclamped phase).

Logistic (or Directed) Belief Net



- Similar to Boltzmann Machine, but with directed, acyclic connections.

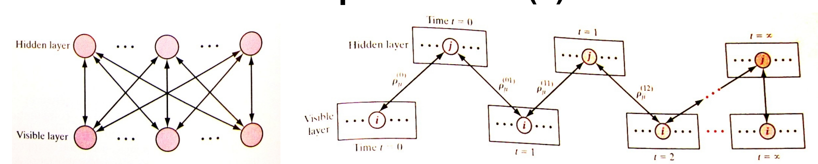
$$P(X_j = x_j | X_1 = x_1, ..., X_{j-1} = x_{j-1}) = P(X_j = x_j | parents(X_j))$$

- Same learning rule:

$$\Delta w_{ji} = \eta \frac{\partial L(\mathbf{w})}{\partial w_{ji}}$$

- With dense connetions, calculation of P becomes intractable.

Deep Belief Net (1)



- Overcomes issues with Logistic Belief Net. Hinton et al. (2006)
- Based on Restricted Boltzmann Machine (RBM): visible and hidden layers, with layer-to-layer full connection but no within-layer connections.
- RBM Back-and-forth update: update hidden given visible, then update visible given hidden, etc., then train $\mathbf{w}$ based on

$$\frac{\partial L(\mathbf{w})}{\partial w_{ji}} = \rho_{ji}^{(0)} - \rho_{ji}^{(\infty)}$$

Deep Belief Net (2)

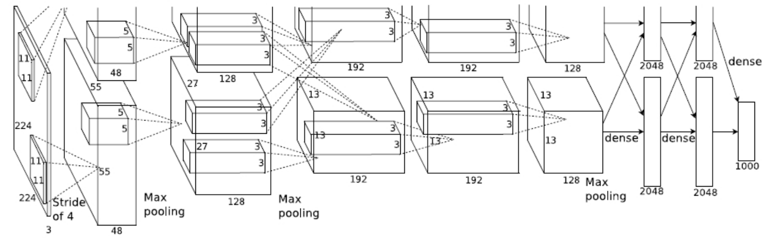
Deep Belief Net = Layer-by-layer training using RBM.

Hybrid architecture: Top layer = undirected, lower layers directed.

1. Train RBM based on input to form hidden representation.
2. Use hidden representation as input to train another RBM.
3. Repeat steps 2-3.

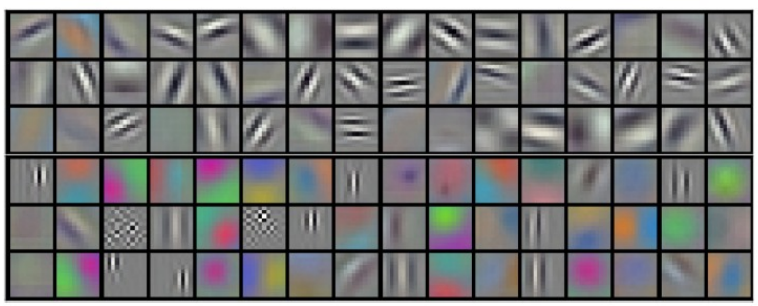
Applications: NIST digit recognition.

Deep Convolutional Neural Networks (1)



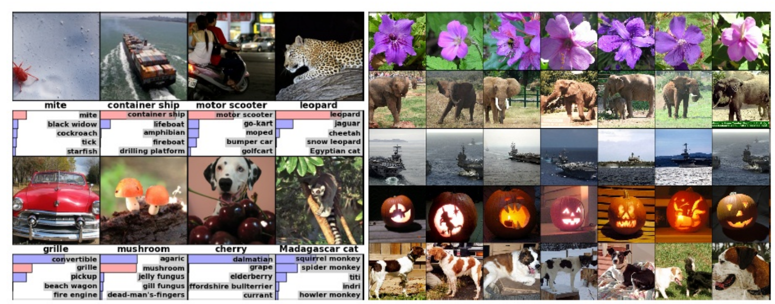
- Krizhevsky et al. (2012)
- Applied to ImageNet competition (1.2 million images, 1,000 classes).
- Network: 60 million parameters and 650,000 neurons.
- Top-1 and top-5 error rates of 37.5% and 17.0%.
- Trained with backprop.

Deep Convolutional Neural Networks (2)



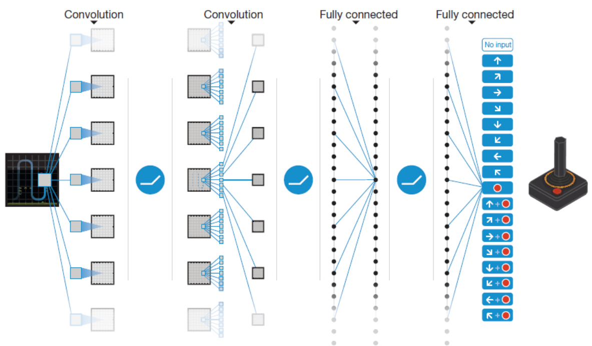
- Learned kernels (first convolutional layer).
- Resembles mammalian RFs: oriented Gabor patterns, color opponency (red-green, blue-yellow).

Deep Convolutional Neural Networks (3)



- Left: Hits and misses and close calls.
- Right: Test (1st column) vs. training images with closest hidden representation to the test data.

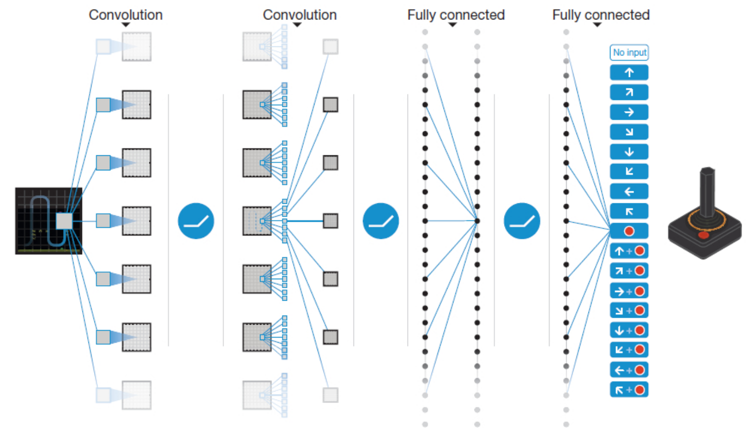
Deep Q-Network (DQN)



Google Deep Mind (Mnih et al. *Nature* 2015).

- Latest application of deep learning to a *reinforcement learning* domain (Q as in Q-learning).
- Applied to *Atari 2600* video game playing.

DQN Overview



- Input: video screen; Output: $Q(s, a)$; Reward: game score.
- $Q(s, a)$: action-value function
  - Value of taking action a when in state s.

DQN Overview

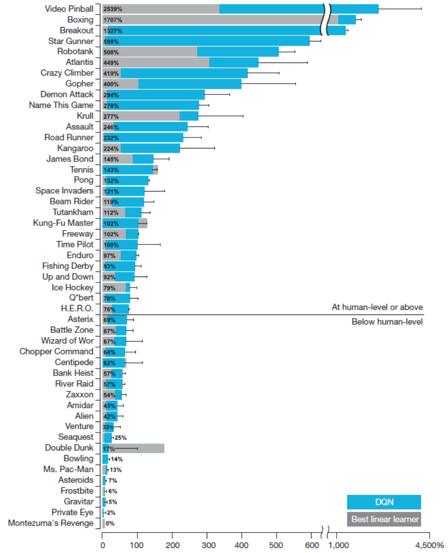
- Input preprocessing
- Experience replay (collect and replay state, action, reward, and resulting state)
- Delayed (periodic) update of Q.
- Moving target $\hat{Q}$ value used to compute error (loss function L, parameterized by weights θ_i).
  - Gradient descent:

$$\frac{\partial L}{\partial \theta_i}$$

DQN Algorithm

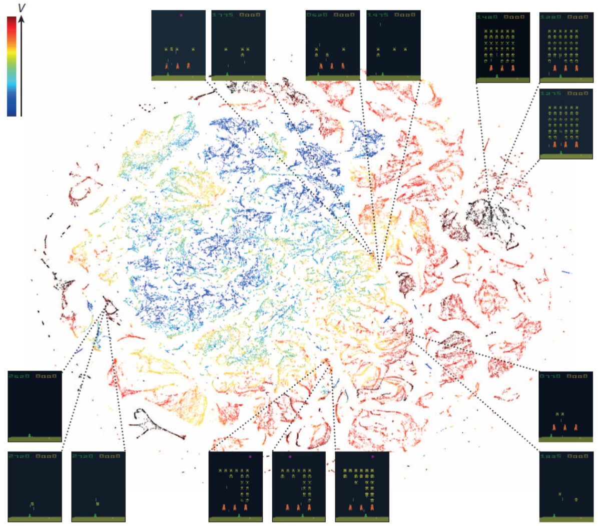
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

DQN Results



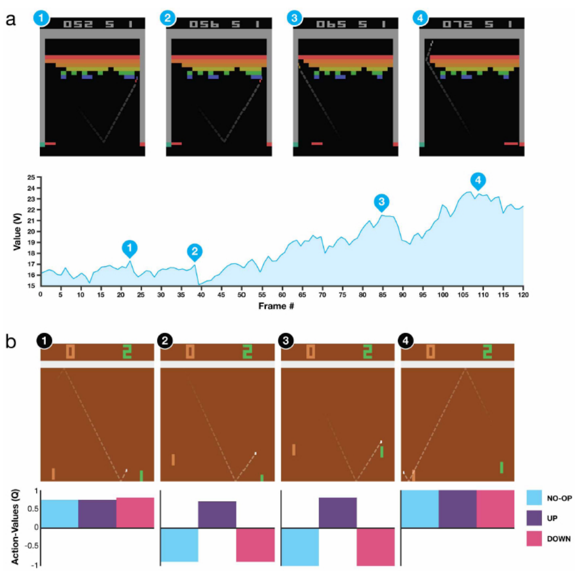
- Superhuman performance on over half of the games.

DQN Hidden Layer Representation (t-SNE map)



- Similar perception, similar reward clustered.

DQN Operation



- Value vs. game state; Game state vs. action value.

Summary

- Deep belief network: Based on Boltzmann machine. Elegant theory, good performance.
- Deep convolutional networks: High computational demand, over the board great performance.
- Deep Q-Network: unique apporach to reinforcement learning. End-to-end machine learning. Super-human performance.