Deep Learning

- training time increases very rapidly with the number of inputs
  - traditionally this means that one must first select (often manually) a small number of features

- deep learning refers to a collection of techniques in which features can also be learned

Deep Learning Network Architecture

- core architecture
  - preprocessing +
    - application dependent (e.g. normalizing)
  - several feature layers +
    - trained by unsupervised learning
  - classical neural network
    - trained by supervised learning

- mimics the structure of the brain
  - sensory organs detect stimuli
  - brain has multiple layers of processing, with higher layers dealing with more abstract levels of information

Stacked Denoising Autoencoders

- an autoencoder learns an identity mapping of its inputs

- contains a single hidden layer
- to avoid overfitting, denoising is used instead of cross-validation to update weights
  - cross-validation = evaluating on a different set of inputs from those used for training
  - denoising = randomly change some of the inputs, then use the distance between the calculated output vector and the original input as the error function

Training a Deep Learning Network

- feature layers are trained first
  - trained by unsupervised learning

- algorithm
  - first feature layer is trained using a stacked denoising autoencoder
    - once trained, the computed weights for first layer are fixed
  - repeat for each additional feature layer
    - train the current layer using a stacked denoising autoencoder
    - the inputs used are the outputs from the previous (already-trained) layer
    - once trained, the computed weights for the current layer are fixed
Training a Deep Learning Network

• once the feature layers are trained, the classical neural network is trained
  – trained by supervised learning
  – inputs are the outputs from the last feature layer
  – feature layer weights can be held constant, or re-trained to fine-tune performance

Variations

• “deep learning” refers to a family of techniques rather than a specific algorithm

• e.g. for the unsupervised learning portion
  – convolutional neural networks with pooling layers
  – deep belief networks with restricted Boltzmann machines
  – clustering via kernel PCA

• e.g. for the supervised learning portion
  – support vector machines (SVMs)

Applications of Neural Networks

• many applications involve pattern recognition, but that is not the only application
  – character recognition
  – electronic nose
  – credit approval
  – stock market prediction
  – image recognition
  – predicting football plays
  – self-driving vehicles
  – handwriting identification
  – handwriting analysis
  – acting on thoughts
  – image compression
  – natural language captioning of photos

Problems with Supervised Learning

Problem: the trained network is no better than its teacher.
Problem: we may not have an expert teacher.
Reinforcement Learning

The agent receives an evaluation of its action (reward or punishment) but is not told the correct action.
- often used when the designer doesn't know how to program the correct actions to take

Challenges.
- **blame attribution problem**
  - reinforcement is often delayed, so must figure out which action(s) were most responsible for the outcome
  - may not be a single action, but a combination of actions carried out in the right circumstances
- the effect of an action also depends on what the agent does subsequently
- **exploit vs explore**
  - exploit utilizes a good solution once it is found, but what if there's something better?
  - too much exploration ignores the benefits of experience and risks missing the best solution altogether

Approaches

- utility-based – agent learns a utility function on states, and uses it to select actions that maximize the expected outcome utility
  - requires agent to know what happens when an action is applied
- Q-learning – agent learns an action-utility function giving the expected utility of taking a given action in a given state
  - does not require the agent to know what happens when an action is applied
  - does not allow lookahead
- reflex – agent learns policy that maps directly from states to actions

Backward Induction

*Backward induction* is the process of reasoning backwards from the end of the problem to find a solution.
Nim

The game:
- two players
- 21 sticks
- players alternate turns, picking up 1, 2, or 3 sticks
- player to pick up last stick wins

Variations have sticks grouped into some number of heaps, or player to pick up last stick loses.

Nim has an optimal strategy.
- first player can guarantee a win if she plays perfectly
- other player can seize on a mistake and turn the tables

Learning Nim via Backward Induction

- learning goal: a direct mapping from state to action

  Initialize a table containing all possible legal moves for each number of sticks.

  Play a game, choosing a move at random from those listed in the table for the current number of sticks.
  - if there are no available moves, a loss is inevitable and game can be abandoned (or pick a random move if a move is required)

  If the computer wins, nothing changes.
  If the computer loses (or reaches state where a loss is inevitable), remove the last move made (to that point) from the table.

Learning Nim via Backward Induction

- exploitation refers to repeating past good choices
- exploration refers to trying new things

How is exploration incorporated into this approach?
- only eliminates known failures
- randomly selects from remaining options

What makes for a good opponent for learning?
- computer needs to experience all the ways to lose as quickly as possible
Learning Nim via Backward Induction

Would this strategy work for other games?
- e.g. tic-tac-toe, checkers, chess, Connect Four, backgammon, ...

Observations.
- this strategy eliminates every move from which we can't force a win
  - it only considers "win" and "not win"
  - blames the last move entirely for a loss
- requires storing every combination of state and action

Temporal Difference

Goal: learn the likelihood of winning from each game state.
- $V(s) > V(t)$ if greater likelihood of winning from state $s$ than from $t$

Playing the game:
- at each state $s$, choose the move leading to the successor state $s'$ with the highest $V(s')$ value

Learning:
- play a game, storing the sequence of states visited
- working backwards, update values of visited states
  $$V(s) \leftarrow V(s) + \alpha [V(s') - V(s)]$$
  - $V(t) = 1$ for a winning state, -1 for a losing state, 0 for a draw
  - learning rate $\alpha$ is the degree to which new information replaces old information
    - $0 =$ consider only old info, $1 =$ consider only new info

$V(s)$ is highest-valued successor

*$s'$ is highest-valued successor