vi

COMP9444: Neural Networks

Applications, Deep Learning Networks

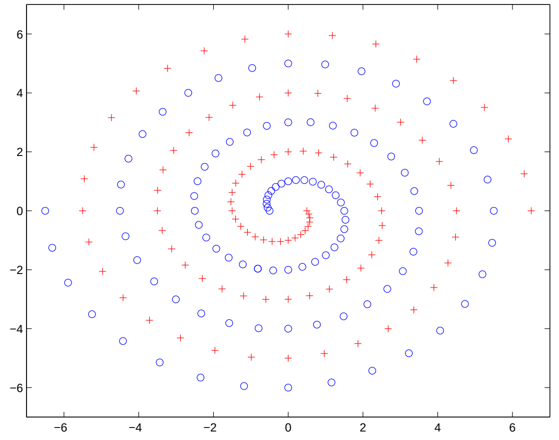
Example Applications

- speech phoneme recognition
- credit card fraud detection
- financial prediction
- image classification
- medical diagnosis
- data mining

Case Studies

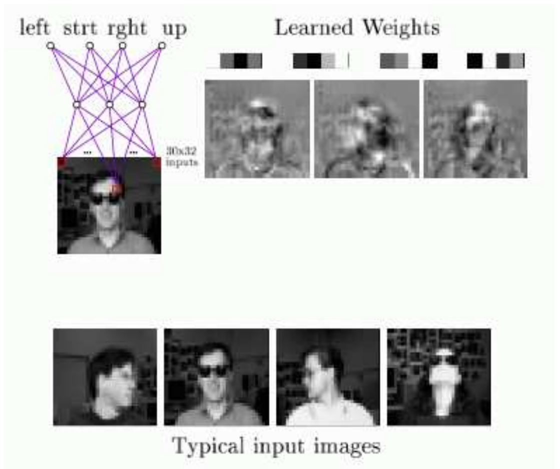
- Twin Spirals
- Face Recognition
- ALVINN
- TD-Gammon

Twin Spirals



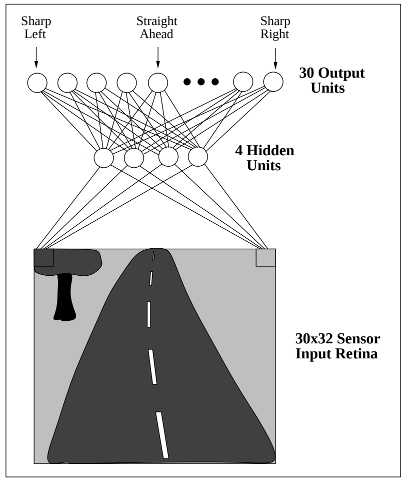
Can be learned with three layers, but not with two layers.

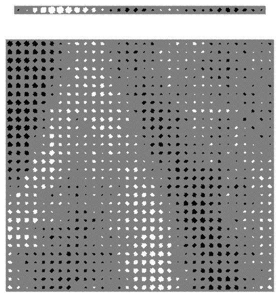
Face Recognition



ALVINN (Pomerleau 1991, 1993)

ALVINN





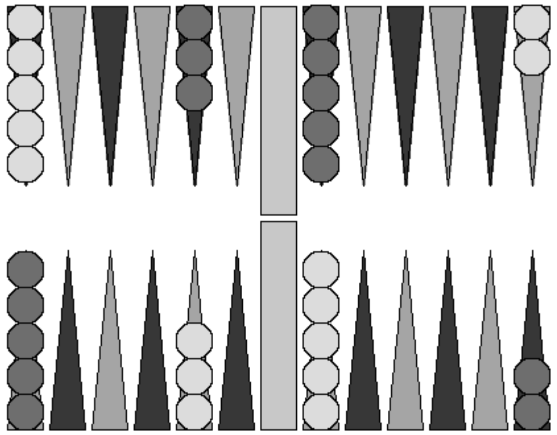
ALVINN

- Autonomous Land Vehicle In a Neural Network
  - $30 \times 32 = 960$ inputs
  - 4 hidden units
  - 30 output units
- later version included a sonar range finder
- centre-of-mass of outputs determines steering direction.
- trained “on the fly” from human driving (behavioural cloning)
- synthetic data generated to cover “emergency” situations
- drove autonomously from coast to coast

ALVINN Training Details

- transformed inputs and outputs also included in training set
  - exposes the network to extreme situations without having to drive off the road.
- trained for two minutes of driving, resulting in 50 real images and $15 \times 50 = 750$ transformed images.
- different networks for dirt roads, city roads, freeways
- able to drive from coast to coast at 70km/h.

Backgammon



Backgammon Neural Network

Two layer neural network

- 196 input units
- 20 hidden units
- 1 output unit

Board encoding

- 4 units $\times$ 2 players $\times$ 24 points
- 2 units for the bar
- 2 units for off the board

The input is the encoded board position,
the output is the value of this position (probability of winning).

Backgammon Play

- how do we play?
  - at each move, roll the dice, find all possible "next board positions", convert them to the appropriate input format, feed them to the network, and choose the one which produces the largest output.
- how do we train the network?
  - by supervised learning (from expert preferences) or by reinforcement learning (from self-play)

Backpropagation

$$w \leftarrow w + \eta(T - P)\frac{\partial P}{\partial w}$$

P	$=$	actual output
T	$=$	target output
w	$=$	weight
η	$=$	learning rate

- How do we choose T ?
  - learn moves from example games?
  - T = final outcome of game? (Widrow-Hoff)
  - Temporal Difference Learning (Sutton)
    (current estimate) $P_k \to \ldots \to P_m \to P_{m+1}$ (final result)

$$T_k = (1-\lambda)\sum_{t=k+1}^{m} \lambda^{t-1-k} P_t + \lambda^{t-k} P_{m+1}$$

TD-Gammon

- Why is TD better than Widrow-Hoff?
  Because it doesn't assign credit indiscriminantly

  ... bad move → good moves → win

- Tesauro trained two networks:
  - EP-network was trained on Expert Preferences
  - TD-network was trained by self play
- TD-network outperformed the EP-network.
- with modifications such as 3-step lookahead and additional hand-crafted input features, TD-Gammon became the best Backgammon player in the world.

Why did it work?

- EP-network is not exposed to extreme situations (similar to ALVINN without transformed images).
- random dice rolls in Backgammon force self-play to explore a much larger part of the search space than it otherwise would.
- humans are bad at probabilistic reasoning?
- other games have been trained by TD-learning, but generally against humans rather than self-play (e.g. Knightcap Chess program).
- genetic algorithm can also produce a surprisingly strong player, but a gradient-based method such as TD-learning is better able to fine-tune the rarely used weights, and exploit the limited nonlinear capabilites of the neural network.

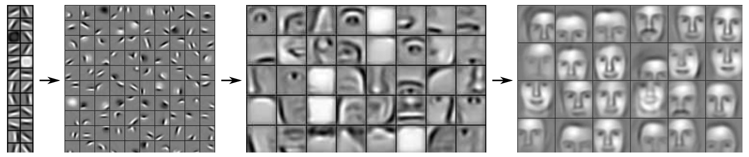
Deep Learning Networks

- Backpropagation using Multi-Layer Perceptrons can be effective for capturing many patterns and relationships, including non-linear properties
- Support Vector Machines can provide even better reliability and generalisation
- There are many limitations of these techniques
  - Typically they are based on hand-engineered features, which requires new features to be developed for new tasks
  - Backpropagation networks require extensive training data, which can be difficult or costly to produce

Deep Learning Networks

- The ability to scale to more complex tasks can be limited (aside from engineered modularity, such as committee machines)
  - With increased depth, training times increase. Learning is less effective as the gradient becomes weaker with depth, as learning must pass down from the classification layer.
  - Training can become stuck in local minima and not find better solutions
- Deep Learning techniques address a number of these issues

Deep Learning Networks



- Representation learning- discovery of features
- Learning from unlabelled data (followed by supervised learning)
- The ability to train deeper networks, and capture intermediate features
- Potential for more modular learning, with re-used features

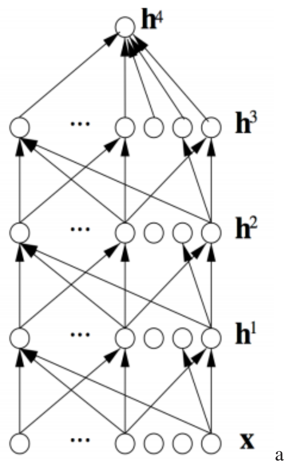
Deep Learning Networks

- Machine Learning theory says we can learn any function with accuracy as close as we want with a single layer, so why bother? 2-layer MLPs and SVMs are “universal”
- The right representation can be much more efficient for particular tasks
- There is significant modularity in the brain- deep networks of re-used features are seen in vision, and are useful for audio and natural language tasks
- A more promising approach for more general AI

Deep Learning Networks

- Common techniques:
  - Unsupervised learning, to pre-train the network
  - Feature learning takes place one layer at a time. Outputs from features of one layer are used as inputs for the next.
  - After pre-training, supervised learning is performed on the network using backpropagation

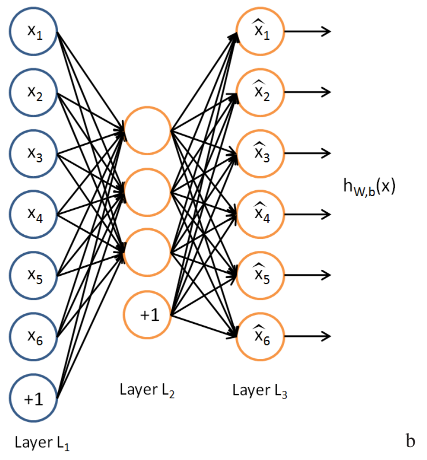
Layer-wise Pre-Training



Deep Learning Networks

- Main approaches:
  - Autoencoder networks (unsupervised pre-training)
  - Restricted Boltzmann Machine networks (unsupervised pre-training)
  - Convolutional Neural Networks (sparse, deep topology)

Autoencoder networks



Autoencoder networks

- Data is provided as input, and the output of the network tries to reconstruct the input
- Learning is performed using backpropagation or related methods
- The target output of the network is set to the input
- The aim of training is to minimise the error of reconstruction
- A reduced set of hidden units is used, creating an information bottleneck

Autoencoder networks

- When used for pre-training, the same weights are used between the input and hidden layer, as between the hidden and output layer

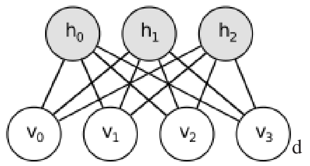
$$W_{input} = W^T_{output}$$

- Reconstruction error is calculated using squared error:

$$E = \frac{1}{2}\|\mathbf{z} - \mathbf{x}\|^2$$

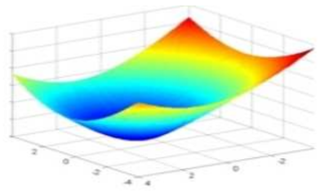
- Training is performed one layer at a time, to build a network of pre-trained features, before using supervised learning.
- The top layer of the network contains output nodes representing classifications

Restricted Boltzmann Machine Networks



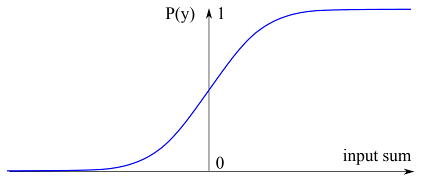
- RBMs are another technique for pre-training, to capture features of the input. They are recurrent networks, with a number of stable states
- Given an input, the network can be sampled. Activations are passed from the input to the hidden layer, then from hidden to the input layer, repeating until stability is reached.
- The visible layer provides a reconstruction of the input. Training the network allows capturing features of the input.

Restricted Boltzmann Machine Networks



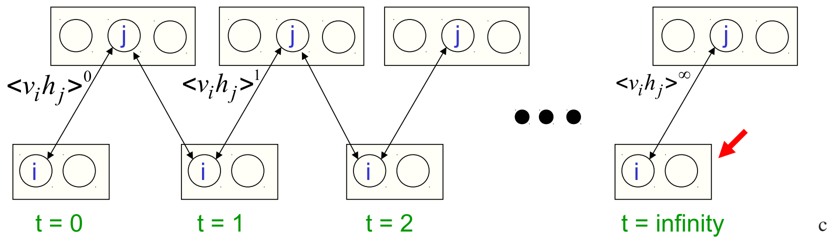
- Stable states of the network have low energy values
- Gibbs sampling is used to find a low energy state
- The energy values of configurations are defined by the weights
- Probabilistic: weights $\rightarrow$ energy values $\rightarrow$ probabilities

Restricted Boltzmann Machine Networks



- Units are binary stochastic neurons. Either on or off, firing is given by a probability value according to the sum of inputs
- The activation function describes the probability of firing $P(y_j)$ as a function of the input $\sum_i x_i w_{ij}$

Alternating Gibbs Sampling



- The input is presented at the visible units
- Hidden units are updated based on probabilistic activations. Visible units are updated subsequently. This repeats until stability is reached.

Learning in RBMs

- The reconstruction reached when the network stabilises is a representation with lower energy than the input.
- We want the network to prefer the input over this "fantasy".
- Energy value of a given configuration: $E(v,h) = -\sum_{i,j} v_ih_jw_{ij}$
- Cost function is given by the difference between the free energy of the configuration with the observed input, and the free energy of the stable state
- Adjust weights according to:

$$\frac{\partial \log P(v)}{\partial w_{ij}} = < v_ih_j >^0 - < v_ih_j >^\infty$$

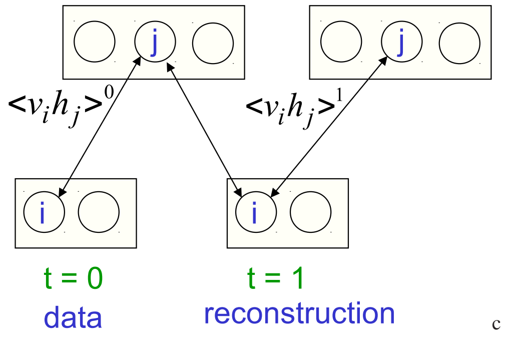
Learning in RBMs

- It takes a lot of time to perform this kind of sampling
- Contrastive Divergence is an approximate method that works well
- Instead of iterating over many steps, perform just one pass. Update visible to hidden, then hidden to visible, then visible to hidden again.
- Adjust weights according to:

$$\Delta w_{ij} = \eta(< v_ih_j >^0 - < v_ih_j >^1)$$

- This does not follow the gradient of the error function directly

Learning in RBMs

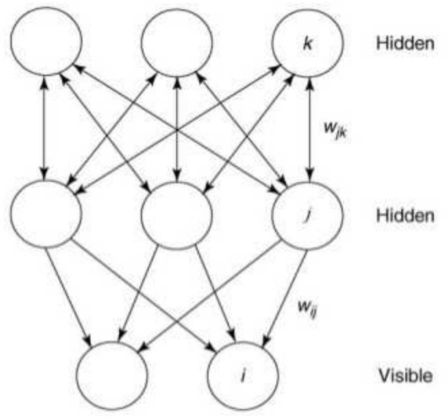


$$\Delta w_{ij} = \eta(< v_ih_j >^0 - < v_ih_j >^1)$$

Learning in RBMs

- Subsequent layers can be learnt in turn, each layer improves the ability of the system to reconstruct the input
- This approach can be used to pre-train a network, before performing supervised learning
- A classification layer can be added to the top layer, representing classes for supervised learning. A fine-tuning stage adjusts weights using an error function defined at the output nodes, by backpropagation.

Learning in RBMs



c

Softmax output

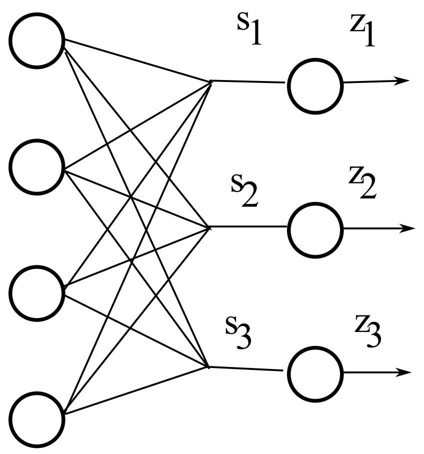
- A common method is to use a Softmax activation function on the output nodes.

$$z_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$$

- Activations are non-local, and represent a probability distribution. The sum of output activations will be 1.
- To perform learning, the following relations are used:

$$\begin{aligned} E &= -\sum_j t_j \log z_j \\ \frac{\partial E}{\partial s_i} &= z_i - t_i \end{aligned}$$

Softmax output

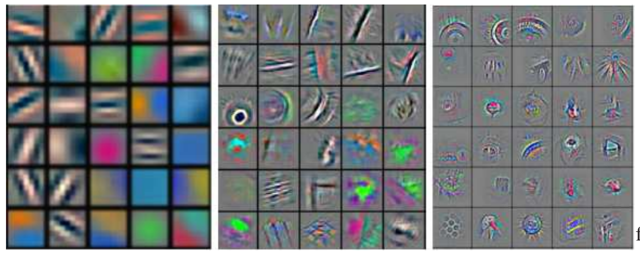


Pre-trained Deep Networks

Putting it all together:

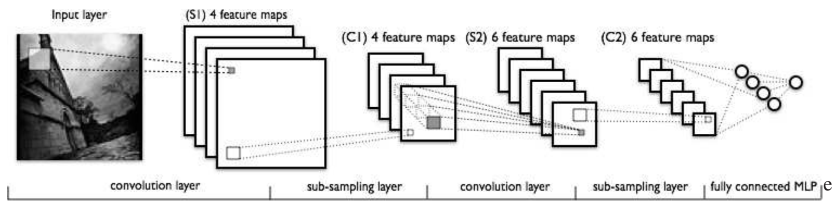
- This approach can be used to pre-train a network, before performing supervised learning
- A classification layer can be added to the top layer, representing classes for supervised learning. A fine-tuning stage adjusts weights using an error function defined at the output nodes, by backpropagation.

Deep Learning Networks



f

Convolutional Neural Networks



Convolutional Neural Networks

- CNNs are a form of deep neural network with a specific topology, based on structure seen in the visual system
- Each unit has a limited receptive field
- Units are convolutional, the same set of weights are used to find a response in multiple positions
- Convolutional and sub-sampling layers perform specific functions, acting in a manner similar to simple and complex cells in the visual system

Summary

- Deep Learning approaches introduce a number of new techniques that allow an increase in depth and modularity of neural networks
- Unsupervised learning allows capturing structure from observations, without relying on feedback from classifications
- Unsupervised pre-training improves the reliability and accuracy of supervised learning
- These techniques offer many new opportunities for machine learning and more general artificial intelligence

Summary

[a] figure by Yoshua Bengio, Montreal. "Learning Deep Architectures for AI"

[b] figure by Andrew Ng, Stanford. "Sparse Autoencoder"

[c] figure by Geoff Hinton, Toronto. "The next generation of neural networks"

[d] figure by LISA lab, Toronto. "Deep Learning tutorials: Restricted Boltzmann Machines". http://deeplearning.net/tutorial/rbm.html

[e] figure by LISA lab, Toronto. "Deep Learning tutorials: Convolutional Neural Networks". http://deeplearning.net/tutorial/lenet.html

[f] figure from Zeiler & Fergus 2013