Neural Networks

Chapter 18, Sec 7, 3rd ed.
Chapter 20, Sec 5, 2nd ed.

Outline

- Brains
- Neural networks
- Perceptrons
- Multilayer perceptrons
- Applications of neural networks
- Discussion

Learning: Neural Networks

- In this topic, we will look at a *nondeclarative* approach in AI.
  - So can't "read off" the meaning of a scheme.
- Idea: Represent *functions* using *networks* of simple arithmetic computing elements.
- These networks will represent functions in the same fashion that circuits represent Boolean functions.
- A network of simple units leads to overall complex behaviour.

Why Neural Networks?

- Major strength: *trainable*.

Why Neural Networks?

- Major strength: *trainable*.
- This area can also be viewed as another approach to *learning*.
  - More accurately, the use of neural networks *requires* that they be trainable.
  - Goal: Learn a function $f(\vec{x}) = y$.

Why Neural Networks?

- Major strength: *trainable*.
- This area can also be viewed as another approach to *learning*.
  - More accurately, the use of neural networks *requires* that they be trainable.
  - Goal: Learn a function $f(\vec{x}) = y$.
- NNs are useful for complex functions with continuous-valued outputs, and a large number of noisy inputs.

Why Neural Networks?

- Major strength: *trainable*.
- This area can also be viewed as another approach to *learning*.
  - More accurately, the use of neural networks *requires* that they be trainable.
  - Goal: Learn a function $f(\vec{x}) = y$.
- NNs are useful for complex functions with continuous-valued outputs, and a large number of noisy inputs.
- Good for applications that are difficult to program directly.
  - E.g. Recognize the number "5"; steer a car.

Why Neural Networks?

- Major strength: *trainable*.
- This area can also be viewed as another approach to *learning*.
  - More accurately, the use of neural networks *requires* that they be trainable.
  - Goal: Learn a function $f(\vec{x}) = y$.
- NNs are useful for complex functions with continuous-valued outputs, and a large number of noisy inputs.
- Good for applications that are difficult to program directly.
  - E.g. Recognize the number "5"; steer a car.
- Another strength: *fault tolerant*.

Motivation

- In trying to build intelligent machines we have one naturally occurring model: the human brain.
    - One way of viewing neural network work is as an attempt to simulate the functioning of the brain on a computer.
    - So these approaches can be considered as dealing with *mathematical models* for the operation of the brain.
    - However these approaches are <u>extremely</u> *limited* compared to the brain.

Motivation

- In trying to build intelligent machines we have one naturally occurring model: the human brain.
  - One way of viewing neural network work is as an attempt to simulate the functioning of the brain on a computer.
  - So these approaches can be considered as dealing with *mathematical models* for the operation of the brain.
  - However these approaches are extremely *limited* compared to the brain.
- In a neural network,
  - the "simple arithmetic computing elements" correspond to *neurons*;
  - the network as a whole corresponds to a collection of interconnected neurons.

Motivation

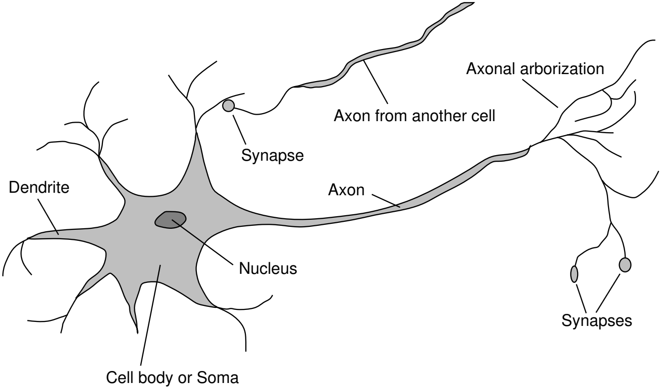
- In trying to build intelligent machines we have one naturally occurring model: the human brain.
  - One way of viewing neural network work is as an attempt to simulate the functioning of the brain on a computer.
  - So these approaches can be considered as dealing with *mathematical models* for the operation of the brain.
  - However these approaches are extremely *limited* compared to the brain.
- In a neural network,
  - the "simple arithmetic computing elements" correspond to *neurons*;
  - the network as a whole corresponds to a collection of interconnected neurons.
- There are many different types of neural networks.
  - We will concentrate on the "feed-forward" network.

Brains

- The exact way in which the brain works is one of the great mysteries of science.
- Fundamental element: The *neuron* or nerve cell.
- Consists of:
  - a body or *soma*,
  - fibres, branching out from the cell body, or *dendrites*,
  - a single long fibre called the *axon*.
- Dendrites branch in a bushy network around the cell, whereas the axon stretches a long distance (about a centimetre but up to a metre).
- The axon also branches into strands that connect to dendrites of other cells via a junction called a *synapse*.

Brains

- 10^{11} neurons of > 20 types, 10^{14} synapses, 1ms–10ms cycle time



Brains

- Signals are propagated from neuron to neuron by an electrochemical reaction.
- Chemical transmitters are released from the synapses and enter the dendrite.
  - These raise or lower the electrical potential of the cell body.
- When the potential reaches a threshold, an electrical pulse is sent down the axon
- This pulse spreads along the branches of the axon, eventually reaching the synapses, and releasing transmitters to the other cells.
- Synapses may be *excitatory* or *inhibitory*.

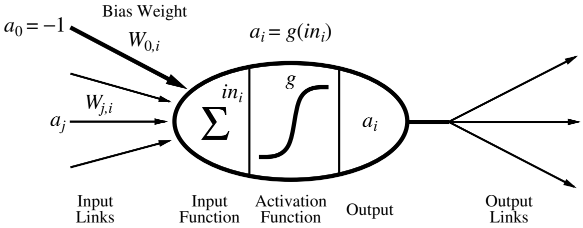
Neural Networks: Architecture

- A NN is made up of nodes or *units* connected by *links*.
- Each link has a numeric *weight* associated with it.
  - Weights are the primary means of long-term storage in NNs.
  - Learning usually takes place by updating the weights.
- Some units are connected to the external environment and serve as *input* or *output* units.
- Each unit:
  - has a set of input links from other units
    + a set of output links to other units.
  - has a current *activation level* or output, and a means of computing the activation level at each step in time, given its inputs and weights.
  - does a *local* computation without the need for global control over the set of units as a whole.
- In practice, most neural networks are implemented in software.

McCulloch–Pitts Unit

- Output is a function of the inputs:

$$a_i \leftarrow g(in_i) = g\left(\Sigma_j W_{j,i} a_j\right)$$



- a_0 is an optional "fixed" input, added for convenience.
- A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

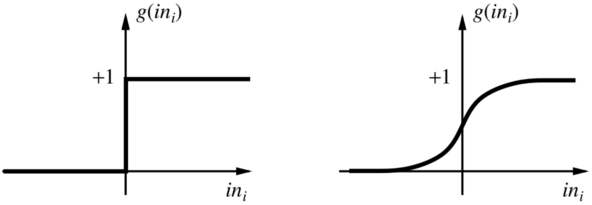
Activation functions

- The activation function g is designed to be "active" (near 1) when the "right" inputs are given, and "inactive" (near 0) when the "right" inputs are given.
- Activation function should be *nonlinear*, since otherwise the network is just a simple linear function.

Activation functions

- The activation function g is designed to be "active" (near 1) when the "right" inputs are given, and "inactive" (near 0) when the "right" inputs are given.
- Activation function should be *nonlinear*, since otherwise the network is just a simple linear function.
- If the activation function is linear then
  - a n-layer network can be shown to be equivalent to a 2-layer network
  - which (as we will see) is very limited as to what it can do.

Activation functions



(a) (b)

(a) is a *step function* or *threshold function*

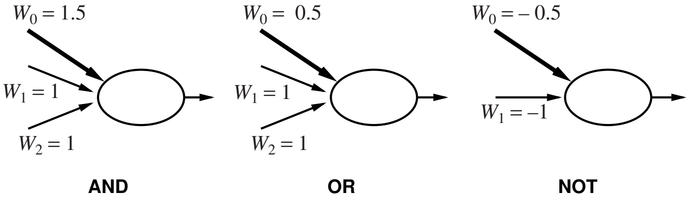
- Outputs 1 when input is +ve; 0 otherwise.

(b) is a *sigmoid* function $1/(1+e^{-x})$

- Changing the bias weight $W_{0,i}$ moves the threshold location

Implementing logical functions

- For a step function transitioning at 0:



- (Recall a_0 is fixed at -1.)
- McCulloch and Pitts: every Boolean function can be implemented

Network structures

- There are a great many kinds of network structures, each of which results in very different computational properties.
- Main distinction: *feed-forward* vs *recurrent* networks.
- Feed-forward networks are DAGs.
- Recurrent networks allow signals to propagate backwards.

Network structures

Feed-forward networks

- *single-layer perceptrons*
- *multi-layer neural networks*

Feed-forward networks implement functions, have no internal state

Network structures

Feed-forward networks

- *single-layer perceptrons*
- *multi-layer neural networks*

Feed-forward networks implement functions, have no internal state

Recurrent networks:

- *Hopfield networks* have symmetric weights ($W_{i,j} = W_{j,i}$)
  - $g(x) = \text{sign}(x)$, $a_i = \pm 1$
  - *holographic associative memory*
- *Boltzmann machines* use stochastic activation functions,
- Recurrent neural nets can have directed cycles with delays

  ☞ Have internal state (like flip-flops), can oscillate etc.

Network structures

- We will deal with feed-forward *layered* networks.
  - The output of a unit is connected only to the inputs of the next layer.
  - No links backwards, nor within the same layer, nor skipping a layer.
- Idea: With no cycles, computation proceeds from input to output units.
- An early hope was that recognition could proceed by:

  sensory inputs → *elementary feature detection*
  → *complex feature detection*
  → *decision making*
  → *(output) actions*

☞ This now seems to be realized in approaches in *deep learning*

Feed-forward neural networks

- Networks are composed of:
  1. *Input units* whose activation value is determined by the environment.
  2. *Output units* whose activation value is an output of the network.
  3. *Hidden units* which lie between input and output units.

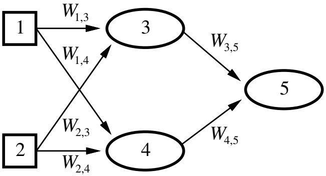
Feed-forward neural networks

- Networks are composed of:
  1. *Input units* whose activation value is determined by the environment.
  2. *Output units* whose activation value is an output of the network.
  3. *Hidden units* which lie between input and output units.
- Networks with no hidden units are called *single layer* networks or *perceptrons*.
  - Otherwise the network is *multilayer*.

Feed-forward neural networks

- Networks are composed of:
  1. *Input units* whose activation value is determined by the environment.
  2. *Output units* whose activation value is an output of the network.
  3. *Hidden units* which lie between input and output units.
- Networks with no hidden units are called *single layer* networks or *perceptrons*.
  - Otherwise the network is *multilayer*.
- We have that:
  - With one (sufficiently large) layer of hidden units, it is possible to represent *any* continuous function of the inputs.
  - With two layers of hidden units, it is possible to represent *any* function (even discontinuous).
  - Note: "represent" $\approx$ "approximate arbitrarily closely".

Feed-forward example

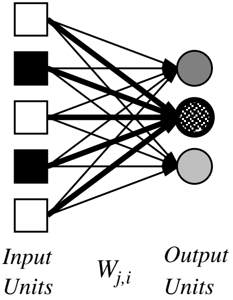


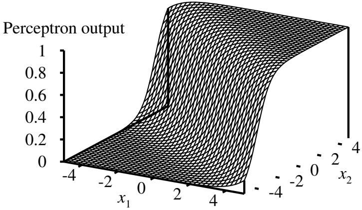
- Feed-forward network = a parameterized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(W_{3,5} \cdot a_3 + W_{4,5} \cdot a_4) \\ &= g(W_{3,5} \cdot g(W_{1,3} \cdot a_1 + W_{2,3} \cdot a_2) + \\ &\qquad W_{4,5} \cdot g(W_{1,4} \cdot a_1 + W_{2,4} \cdot a_2)) \end{aligned}$$

- Adjusting weights changes the function:
  - do learning this way!

Single-layer networks: Perceptrons





- Output units all operate separately – no shared weights
  ☞ So we can limit our analysis to a single output unit.
- Adjusting weights moves the location, orientation, and steepness of cliff

Expressiveness of perceptrons

- Consider a perceptron with $g =$ step function.
  - Can represent AND, OR, NOT, majority, etc.

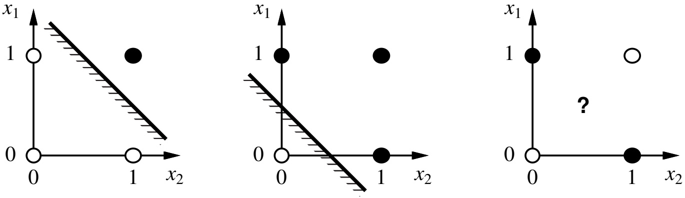
Expressiveness of perceptrons

- Consider a perceptron with $g =$ step function.
  - Can represent AND, OR, NOT, majority, etc.
  - Can't represent XOR

Expressiveness of perceptrons

- Consider a perceptron with g = step function.
  - Can represent AND, OR, NOT, majority, etc.
  - Can't represent XOR
  - Represents a *linear separator* (or hyperplane) in input space:

$$\Sigma_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) x_1 **and** x_2 (b) x_1 **or** x_2 (c) x_1 **xor** x_2

The Limits of Perceptrons

- We can, e.g. represent the *majority function*, which outputs 1 if more than half of its inputs are 1.

The Limits of Perceptrons

- We can, e.g. represent the *majority function*, which outputs 1 if more than half of its inputs are 1.
  - Use a perceptron with each $W_j = 1$ and threshold $t = n/2$.

The Limits of Perceptrons

- We can, e.g. represent the *majority function*, which outputs 1 if more than half of its inputs are 1.
  - Use a perceptron with each $W_j = 1$ and threshold $t = n/2$.
  - This would require a decision tree with $O(2^n)$ nodes. (Why?)

The Limits of Perceptrons

- We can, e.g. represent the *majority function*, which outputs 1 if more than half of its inputs are 1.
  - Use a perceptron with each $W_j = 1$ and threshold $t = n/2$.
  - This would require a decision tree with $O(2^n)$ nodes. (Why?)
- However, perceptrons are severely limited, in that they can only represent *linearly separable* functions.
- XOR, for example, is not linearly separable.

Learning Linearly Separable Functions

- The (relatively) good news is that:
  - There is a perceptron algorithm that will learn any linearly separable function, given enough training examples.

Learning Linearly Separable Functions

- The (relatively) good news is that:
  - There is a perceptron algorithm that will learn any linearly separable function, given enough training examples.
- The perceptron learning method (as with most NN learning algorithms) follows a gradient descent (i.e. hill climbing!) scheme.
  - The initial network has randomly assigned edge weights.
  - The network is then updated to try to make it consistent with examples.
    - This is done by making small adjustments between the observed and predicted values.
  - The update phase is repeated some number of times.
    - Each such complete run through the examples is called an *epoch*.

Perceptron Learning

- Learn by adjusting weights to reduce *error* on training set
- For an example, if the predicted output is O and correct output is T, then the error is given by $Err = T - O$.
    - If Err is $+$ve we need to increase O, and decrease if $-$ve.
- Now, each input unit j contributes $W_j \times x_j$ to the total input.
- So if x_j is $+$ve, an increase in W_j will tend to increase O, and vice versa.

Perceptron Learning

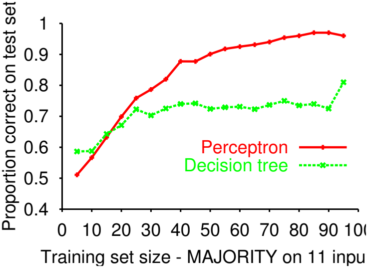
- Learn by adjusting weights to reduce *error* on training set
- For an example, if the predicted output is O and correct output is T, then the error is given by $Err = T - O$.
  - If Err is $+$ve we need to increase O, and decrease if $-$ve.
- Now, each input unit j contributes $W_j \times x_j$ to the total input.
- So if x_j is $+$ve, an increase in W_j will tend to increase O, and vice versa.
- We can achieve this with the *perceptron learning rule*:

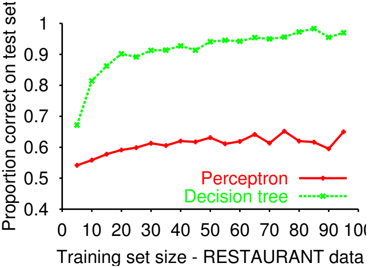
$$W_j \leftarrow W_j + \alpha \times x_j \times Err.$$

- α is called the *learning rate*, and is determined empirically.
  - If α is too large it will "overshoot"
  - If α is too small, the perceptron will converge too slowly.
- If $Err = 0$ then W_j is unchanged.

Perceptron learning contd.

- The perceptron learning rule converges to a consistent function for any linearly separable data set





- Perceptron learns majority function easily; DTL is hopeless
- DTL learns restaurant function easily; perceptron is hopeless

Perceptrons: Summary

- The *perceptron convergence theorem* guarantees that:

  > *the learning method will find a solution state, and will converge to a set of weights that correctly classifies the examples,*

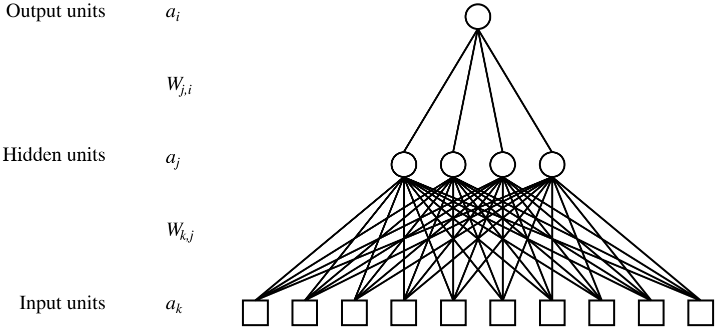
  provided that:

  > *the examples represent a linearly separable function.*

- This created a lots of excitement when it was announced.
  - Here was a device that resembled a neuron, was simple, and could correctly learn any representable function!
- It was not until 1969 that Minsky and Papert took what should have been the first step:
  - analyse the class of linearly representable functions and show their limitations.

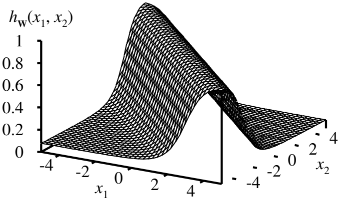
Multilayer Feed-Forward Neural Networks

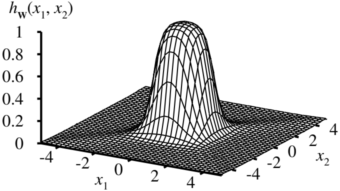
- Layers are usually fully connected.
- Numbers of *hidden units* typically chosen by hand.



Expressiveness of MLPs

- Can represent all continuous functions with 2 layers, all functions with 3 layers (including discontinuous functions).





Learning in Feed-forward Networks

- Most early work was concentrated on single-layer perceptrons.
  - Problem: updating weights between the hidden units and the inputs.
  - Although an error term can be calculated for the outputs, it was not clear how to do so for the hidden units.
- To date learning algorithms for multilayer networks are neither efficient nor guaranteed to converge to a global optimum.
  - Changing with *deep learning*
  - *Learning* is essential, since programming by hand is infeasible
- The most popular method for leaning in multilayer networks is called *back-propagation*.
- Back-propagation has been around since 1969, but was essentially ignored, then re-discovered in the mid-1980s.

Back-Propagation Learning

- Assume that
  - the network is fully connected,
  - there is only 1 hidden layer, and
  - the number of layers ($2 +$ input) and units is set in advance.
  - ☞ In general determining the number of hidden units is difficult.

Back-Propagation Learning

- Assume that
  - the network is fully connected,
  - there is only 1 hidden layer, and
  - the number of layers (2 + input) and units is set in advance.
  - ☞ In general determining the number of hidden units is difficult.
- Learning proceeds in much the same way as for a perceptron:
  - Example inputs are presented to the network
  - If the network computes the correct output, nothing is done.
  - If there is an error, the weights are adjusted to reduce this error.
    - Key: Assess blame and divide it among the contributing weights.
    - Problem: Many edges connect an input to an output. (In a perceptron there is only one.)

Back-Propagation Learning

- For the output layer, the weight update rule is the same as before except:
  - the activation value of the hidden unit a_j is used instead of the input value, and
  - the rule contains a term for the *gradiant* of the activation function.

Updating Output Units

- If Err_i is the error $(T_i - O_i)$ at output node a_i, then the weight update rule for the link from unit j to i is given by:

  $$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i \times g'(in_i).$$

  where:

  - g' is the derivative of the activation function g
  - in_i is the weighted sum of inputs to unit i.
  - a_j is the output value of unit j.
  - α is the learing rate.

- For convenience the weight update function is expressed using a new error term Δ_i which for output nodes is given by:

  $$\Delta_i = Err_i \times g'(in_i).$$

- The update rule then is: $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$.

Updating Hidden Units

- We need an error term for the edges between input units and hidden units.
- Intuitively the error assigned to a hidden unit a_j should depend on
  - the errors of the units that use its output, and
  - the state of the unit's own activation.
- So for hidden unit a_j, the total error is the weighted sum of the errors of the units that use a_j's output.
- That is, the error for unit a_j is "back propagated" by:

$$\Delta_j = g'(in_j) \times \sum_i (W_{j,i} \times \Delta_i).$$

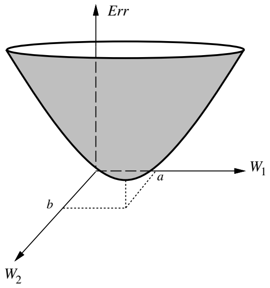
- $g'(in_j)$ is highest for values of inputs close to the threshold.
  - Thus units close to their threshold (on those inputs) will assume more responsibility for the overall error of the system.

Updating Hidden Units (Concluded)

- Once the errors have been computed, the weight update rule can be applied.
- This rule is almost the same as the rule for the output layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j.$$

Back-Propagation as Gradient Descent



- Weight updating can be seen as *gradient descent* on the error surface.
- Current values of W_1 and W_2 define a point on this surface.
- When $W_1 = a$ and $W_2 = b$, the error is minimized.
- We take the slope of the surface along the axis formed by each weight.
  - ☞ I.e. approximate the *partial derivative*

Arbitrary Multi-Layer Networks: Algorithm Summary

- For each example:
  - Compute the Δ (error) values for the output units using the observed error.
  - Starting with the output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached:
    - Propagate the Δ values back to the previous layer.
    - Update the weights between the two layers.
- This algorithm is run on each *epoch* until the network has converged or until some other stopping criterion is met.

Back-Propagation Learning: Summary

- Output layer: (nearly) the same as for single-layer perceptron,

  $$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i \text{ where } \Delta_i = Err_i \times g'(in_i)$$

- Hidden layer: *back-propagate* the error from the output layer:

  $$\Delta_j = g'(in_j) \times \sum_i W_{j,i} \Delta_i \ .$$

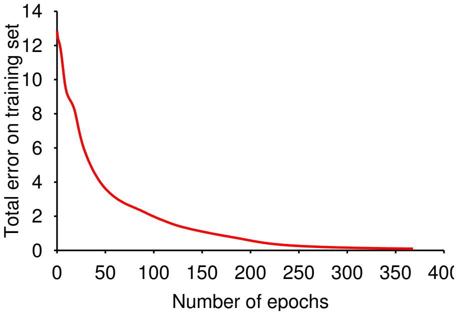
- Update rule for weights in hidden layer:

  $$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j \ .$$

- See the text for the derivation of these equations

Back-propagation learning contd.

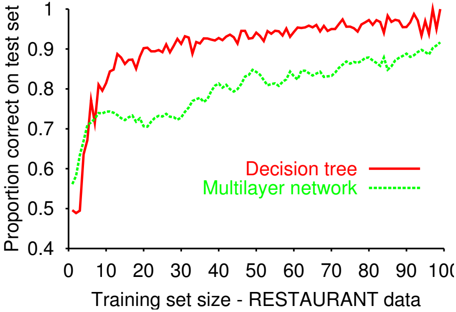
- *Training curve* for 100 restaurant examples: finds a near-exact fit



- Typical problems: slow convergence, local minima

Back-propagation learning contd.

- Learning curve for MLP with 4 hidden units:



- MLPs are quite good for complex pattern recognition tasks, but output classifications cannot be understood easily
- This makes MLPs ineligible for tasks such as credit card and loan approvals, where law requires clear unbiased criteria

Network structure

- So far we've just dealt with networks with a fixed structure.
- A problem is how to select a network *topology*.
  - If a network is too small, the model will be unable to represent the desired function.
  - If too large, the network will be able to memorize the examples, but won't generalise well.
    - As in statistical models, NNs are subject to *overfitting*.
- Another problem is that the number of units in a hidden layer may grow exponentially with the inputs.
  - To date there is no good theory characterising functions that can be represented by a small number of units.

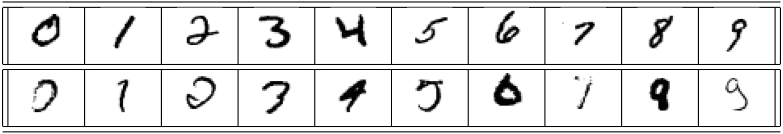
Network structure

- Finding a good network structure can be seen as a search problem over the space of network structures.
- This is a very large space, and evaluating a state means running the whole network-training protocol.
  - So, very expensive.
- One approach is *optimal brain damage*:
  - Remove weights from an initially fully-connected network.
- Another approach is to try to *grow* a network from a smaller one.

Applications

- There have been many significant applications of neural networks.
- In each case, the network design was the result of months of trial-and-error experimentation by researchers.
- Moral: NNs cannot magically solve problems without thought on the part of the network designer.

Application: Handwritten digit recognition



- 3-nearest-neighbor $= 2.4\%$ error
  - ☞ Compare against 60,000 images
- 400–300–10 unit MLP $= 1.6\%$ error
- LeNet: 768–192–30–10 unit MLP $= 0.9\%$ error
- Current best $< 0.3\%$ error (comparable to humans)

Summary

- Perceptrons (one-layer networks) insufficiently expressive
- Multi-layer networks are sufficiently expressive; can be trained by gradient descent, i.e., error back-propagation
- Many applications: speech, driving, handwriting, fraud detection, etc.
- Engineering, cognitive modelling, and neural system modelling subfields have largely diverged

Discussion: Deep Learning

See: *Deep Learning: A Critical Appraisal*, by Gary Marcus, NYU

Overview

- The ideas behind deep learning (DL) have been around for $\approx$ 40 years, but it is in the last 5 years that it has taken off.
- This in part is due to increased computational power and data sets.
- DL has had many very impressive successes
- However, it is important to distinguish the things that DL can and can't do.

What is DL?

Marcus: DL is

> *...essentially a statistical technique for classifying patterns, based on sample data, using neural networks with multiple layers*

- The NNs in DL are most often multi-layer feed-forward networks, as we've seen, using back-propagation for learning.
- "deep" = several hidden layers

DL Networks

- Most DL networks make heavy use of *convolution* that captures a notion of *translational invariance*
  - I.e. if you move an object around, it remains the same object.
- Good for self-generating intermediate representations,
  - e.g. things like horizontal lines or other elements of picture structure.
- One issue: Local minima
  - However techniques have been developed for getting out of a local minimum

Applications

- Broadly: classification system.
  - The goal is typically to decide which category (defined by the output units on the neural network) a given input belongs to.
- Examples:

  Speech sounds $\Rightarrow$ set of labels (e.g. words or phonemes)

  Set of images $\Rightarrow$ a set of labels (e.g. pictures of cars are labeled as cars)

  Pixels $\Rightarrow$ joystick positions (in DeepMind's Atari game system)
- In the classic DL paper (Krizhevsky, Sutskever, & Hinton, 2012), a nine layer neural network with 60 million parameters and 650,000 nodes was trained on roughly a million distinct images drawn from approximately one thousand categories

Challenges faced by DL systems

- Good for *interpolation*, less so for *extrapolation*
  - I.e. good when there is a close fit with training and classification instances.
  - Good for problems that are self-contained and don't need broad general knowledge.
  - but problematic in attempting to move a plan to a new environment

Challenges faced by DL systems

- Good for *interpolation*, less so for *extrapolation*
  - I.e. good when there is a close fit with training and classification instances.
  - Good for problems that are self-contained and don't need broad general knowledge.
  - but problematic in attempting to move a plan to a new environment
- Learning is often brittle, easily fooled.
  - E.g. misclassifying a traffic sign as a refrigerator

Challenges faced by DL systems

- Good for *interpolation*, less so for *extrapolation*
  - I.e. good when there is a close fit with training and classification instances.
  - Good for problems that are self-contained and don't need broad general knowledge.
  - but problematic in attempting to move a plan to a new environment
- Learning is often brittle, easily fooled.
  - E.g. misclassifying a traffic sign as a refrigerator
- Unable to deal with structure
  - E.g. a sentence is seen as a string of words, and not composed of a recursive phrase structure.

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)
- Reasoning. E.g.
  - How to fix a bicycle with a rope caught in its spokes.

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)
- Reasoning. E.g.
  - How to fix a bicycle with a rope caught in its spokes.
  - Commonsense knowledge:

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)
- Reasoning. E.g.
  - How to fix a bicycle with a rope caught in its spokes.
  - Commonsense knowledge:

    Mozart visited Vienna 3 times

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)
- Reasoning. E.g.
  - How to fix a bicycle with a rope caught in its spokes.
  - Commonsense knowledge:

    Mozart visited Vienna 3 times

    He died in Vienna

Issues with symbolic reasoning

- Can't be "told" new information
  - e.g. "*schmister* is a sister between the ages of 10 and 21." People can immediately deal with this; a NN can't
- A NN is essentially a black box. Can't explain or defend a decision, e.g. in medical diagnosis, or getting a loan from a bank
- DL as yet lacks the incrementality, transparency and debuggability of classical programming (Peter Norvig)
- Reasoning. E.g.
  - How to fix a bicycle with a rope caught in its spokes.
  - Commonsense knowledge:

    Mozart visited Vienna 3 times
    He died in Vienna
    On which visit did he die?

Discussion

- DL is an approach for
  - optimizing a complex system

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.
- Excellent at solving closed-end classification problems, where
  - a wide range of signals is to be mapped onto a limited number of categories,

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.
- Excellent at solving closed-end classification problems, where
  - a wide range of signals is to be mapped onto a limited number of categories,
  - given sufficient data and where the test set closely resembles the training set.

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.
- Excellent at solving closed-end classification problems, where
  - a wide range of signals is to be mapped onto a limited number of categories,
  - given sufficient data and where the test set closely resembles the training set.
- Work less well when
  - there are limited amounts of training data or

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.
- Excellent at solving closed-end classification problems, where
  - a wide range of signals is to be mapped onto a limited number of categories,
  - given sufficient data and where the test set closely resembles the training set.
- Work less well when
  - there are limited amounts of training data or
  - when the test set differs importantly from the training set, or

Discussion

- DL is an approach for
  - optimizing a complex system
  - that represents a mapping between inputs and outputs,
  - given a sufficiently large data set.
- Excellent at solving closed-end classification problems, where
  - a wide range of signals is to be mapped onto a limited number of categories,
  - given sufficient data and where the test set closely resembles the training set.
- Work less well when
  - there are limited amounts of training data or
  - when the test set differs importantly from the training set, or
  - when the space of examples is broad and filled with novelty.

Risks to the field of AI

Marcus mentions two possible risks:

- The potential of another "AI winter", if results fall short of the hype.
  - Possibly DL research is approaching a "wall"
- Is AI research getting trapped in a "local minimum"?

  I.e. focussing too much on just one part of AI,

  - focusing too much on a particular class of accessible but limited models, and
  - neglecting possibly riskier areas that might eventually lead to more significant results.

How to proceed?

- More focus on unsupervised learning, experimentation

How to proceed?

- More focus on unsupervised learning, experimentation
- Hybrid (mixed symbolic and non-symbolic) models

How to proceed?

- More focus on unsupervised learning, experimentation
- Hybrid (mixed symbolic and non-symbolic) models
- Other NN models, e.g. recurrent approaches

How to proceed?

- More focus on unsupervised learning, experimentation
- Hybrid (mixed symbolic and non-symbolic) models
- Other NN models, e.g. recurrent approaches
- More ambitious challenges