An Introduction to Neural Networks

Vincent Cheung

Kevin Cannons

Signal & Data Compression Laboratory

Electrical & Computer Engineering

University of Manitoba

Winnipeg, Manitoba, Canada

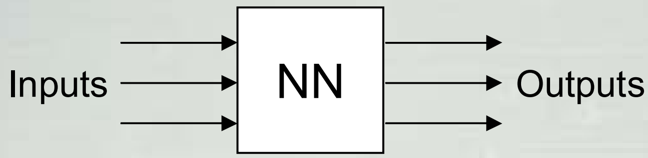
Advisor: Dr. W. Kinsner

Outline

- Fundamentals
- Classes
- Design and Verification
- Results and Discussion
- Conclusion

What Are Artificial Neural Networks?

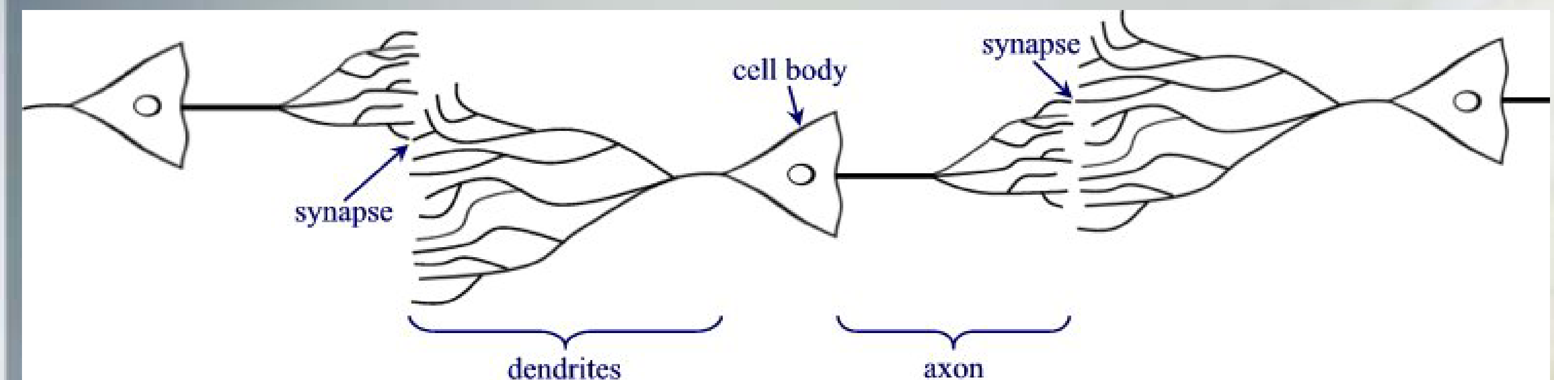
- An extremely simplified model of the brain
- Essentially a function approximator
  - Transforms inputs into outputs to the best of its ability

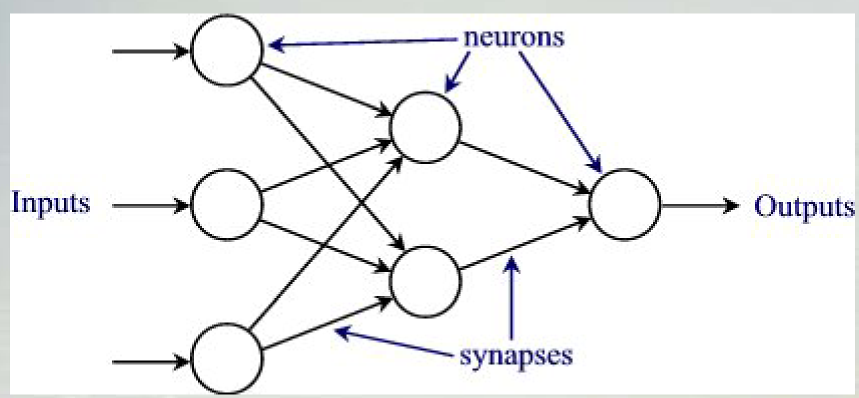


Fundamentals
Classes
Design
Results

What Are Artificial Neural Networks?

- Composed of many “neurons” that co-operate to perform the desired function





What Are They Used For?

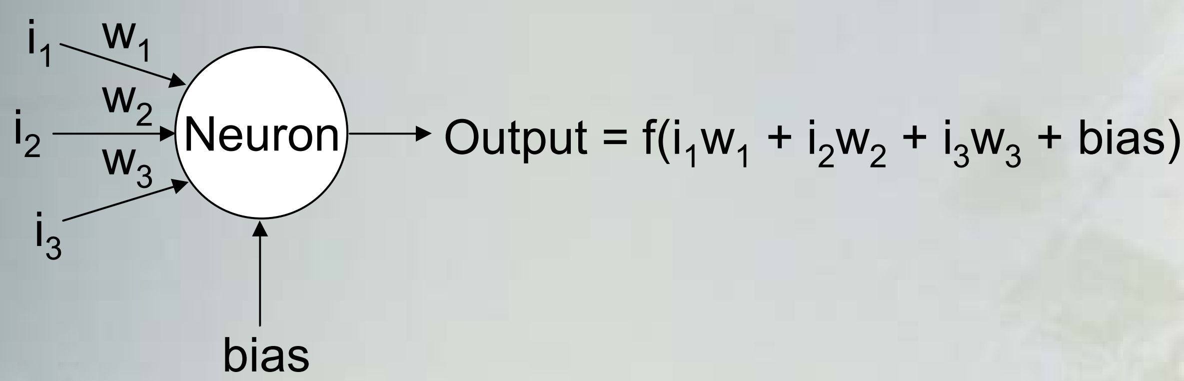
- Classification
  - Pattern recognition, feature extraction, image matching
- Noise Reduction
  - Recognize patterns in the inputs and produce noiseless outputs
- Prediction
  - Extrapolation based on historical data

Why Use Neural Networks?

- Ability to learn
  - NN’s figure out how to perform their function on their own
  - Determine their function based only upon sample inputs
- Ability to generalize
  - i.e. produce reasonable outputs for inputs it has not been taught how to deal with

How Do Neural Networks Work?

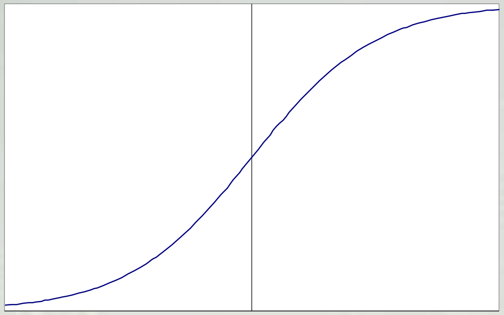
- The output of a neuron is a function of the weighted sum of the inputs plus a bias



- The function of the entire neural network is simply the computation of the outputs of all the neurons
  - An entirely deterministic calculation

Activation Functions

- Applied to the weighted sum of the inputs of a neuron to produce the output
- Majority of NN's use sigmoid functions
  - Smooth, continuous, and monotonically increasing (derivative is always positive)
  - Bounded range - but never reaches max or min
    - Consider "ON" to be slightly less than the max and "OFF" to be slightly greater than the min



Activation Functions

- The most common sigmoid function used is the logistic function
  - $f(x) = 1/(1 + e^{-x})$
  - The calculation of derivatives are important for neural networks and the logistic function has a very nice derivative
    - $f'(x) = f(x)(1 - f(x))$
- Other sigmoid functions also used
  - hyperbolic tangent
  - arctangent
- The exact nature of the function has little effect on the abilities of the neural network

Where Do The Weights Come From?

- The weights in a neural network are the most important factor in determining its function
- Training is the act of presenting the network with some sample data and modifying the weights to better approximate the desired function
- There are two main types of training
  - Supervised Training
    - Supplies the neural network with inputs and the desired outputs
    - Response of the network to the inputs is measured
      - The weights are modified to reduce the difference between the actual and desired outputs

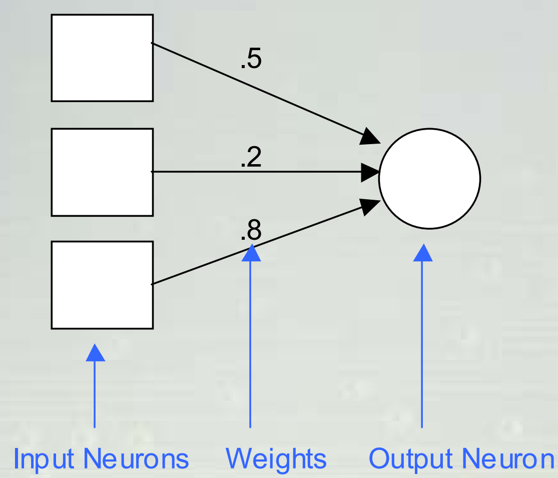
Where Do The Weights Come From?

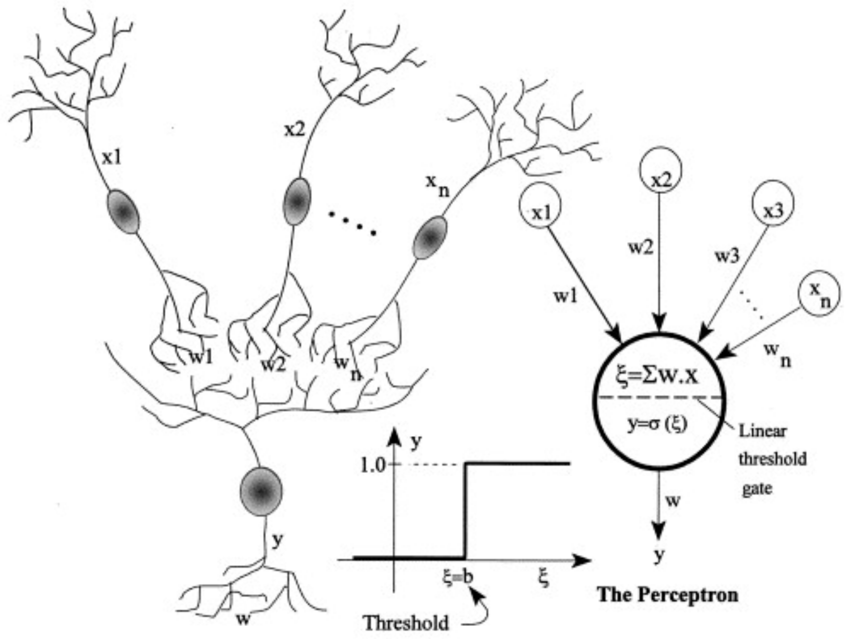
- Unsupervised Training
  - Only supplies inputs
  - The neural network adjusts its own weights so that similar inputs cause similar outputs
    - The network identifies the patterns and differences in the inputs without any external assistance

- Epoch
  - One iteration through the process of providing the network with an input and updating the network's weights
  - Typically many epochs are required to train the neural network

Perceptrons

- First neural network with the ability to learn
- Made up of only input neurons and output neurons
- Input neurons typically have two states: ON and OFF
- Output neurons use a simple threshold activation function
- In basic form, can only solve linear problems
  - Limited applications





x1
x2
x n
w1
w2
w n
y
w
y
1.0
ξ=b
ξ
Threshold
x1
x2
x3
x n
w1
w2
w3
w n
ξ=Σw.x
y=σ (ξ)
Linear threshold gate
w
y
The Perceptron

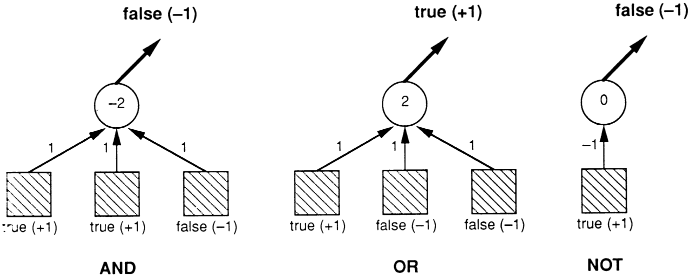


Figure 2.2
AND, OR, and NOT functions computed by single-cell linear discriminant models

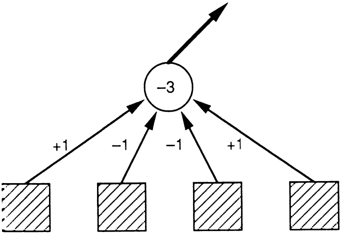
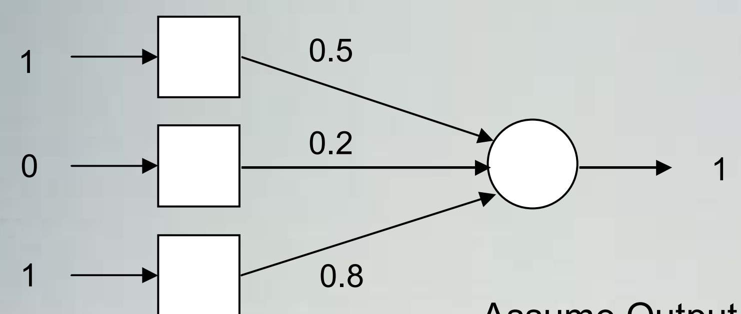


Figure 2.3
A selector cell for the inputs $\langle +1\ \ -1\ \ -1\ \ +1 \rangle$.

How Do Perceptrons Learn?

- Uses supervised training
- If the output is not correct, the weights are adjusted according to the formula:
  - $w_{new} = w_{old} + \alpha(\text{desired} - \text{output})*\text{input}$

α is the learning rate



1 * 0.5 + 0 * 0.2 + 1 * 0.8 = 1.3

Assuming Output Threshold = 1.2

1.3 > 1.2

Assume Output was supposed to be 0
→ update the weights

Assume α = 1

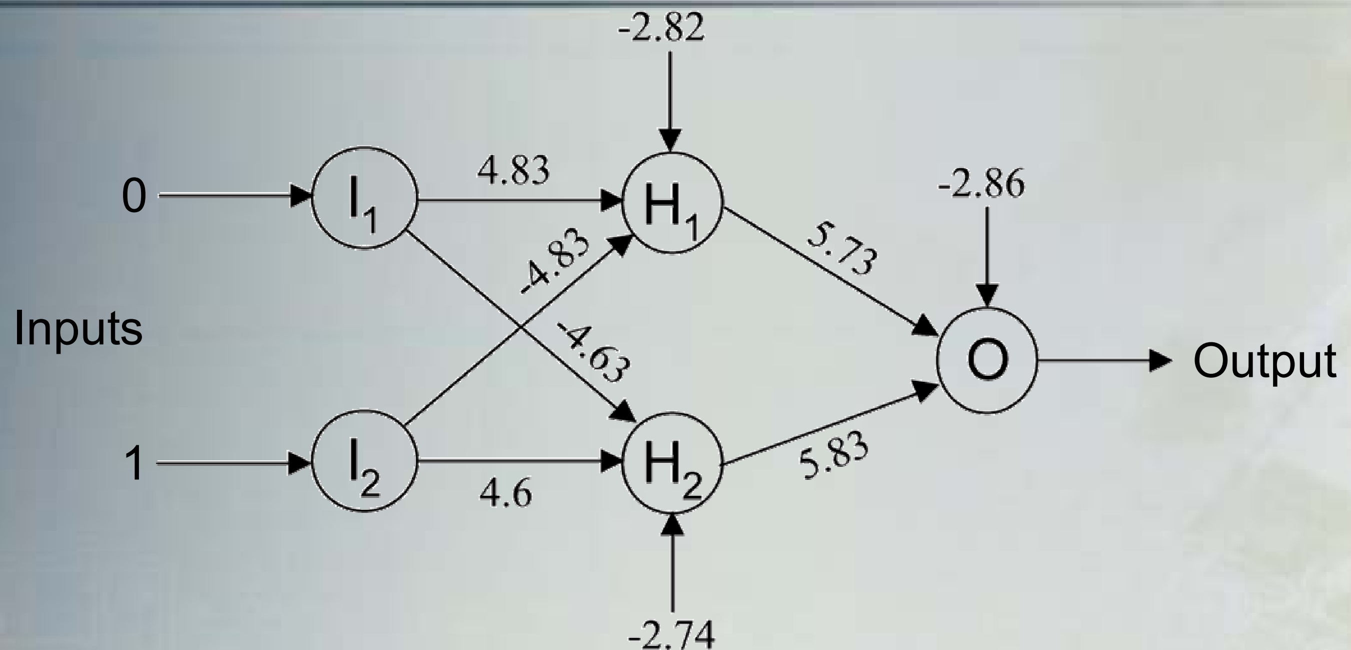
$W_{1new} = 0.5 + 1*(0-1)*1 = -0.5$
$W_{2new} = 0.2 + 1*(0-1)*0 = 0.2$
$W_{3new} = 0.8 + 1*(0-1)*1 = -0.2$

Multilayer Feedforward Networks

- Most common neural network
- An extension of the perceptron
  - Multiple layers
    - The addition of one or more “hidden” layers in between the input and output layers
  - Activation function is not simply a threshold
    - Usually a sigmoid function
  - A general function approximator
    - Not limited to linear problems
- Information flows in one direction
  - The outputs of one layer act as inputs to the next layer

Fundamentals
Classes
Design
Results

XOR Example



Inputs: 0, 1

H_1: Net = 0(4.83) + 1(-4.83) – 2.82 = -7.65
Output = 1 / (1 + $e^{7.65}$) = 4.758 x 10^{-4}

H_2: Net = 0(-4.63) + 1(4.6) – 2.74 = 1.86
Output = 1 / (1 + $e^{-1.86}$) = 0.8652

O: Net = 4.758 x 10^{-4}(5.73) + 0.8652(5.83) – 2.86 = 2.187
Output = 1 / (1 + $e^{-2.187}$) = 0.8991 ≡ “1”

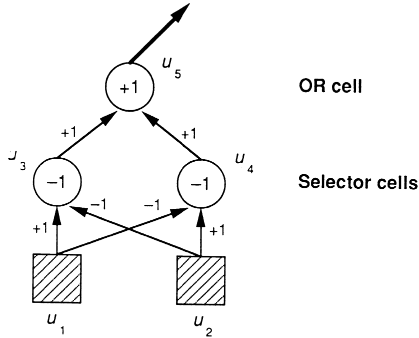


Figure 2.6
Flat network for computing XOR using selector ce

Backpropagation

- Most common method of obtaining the many weights in the network
- A form of supervised training
- The basic backpropagation algorithm is based on minimizing the error of the network using the derivatives of the error function
  - Simple
  - Slow
  - Prone to local minima issues

Fundamentals
Classes
Design
Results

Backpropagation

- Most common measure of error is the mean square error:

$$E = (\text{target} - \text{output})^2$$

- Partial derivatives of the error wrt the weights:
  - Output Neurons:

let: $\delta_j = f'(net_j)\,(target_j - output_j)$
$\partial E/\partial w_{ji} = -output_i\ \delta_j$

j = output neuron
i = neuron in last hidden

  - Hidden Neurons:

let: $\delta_j = f'(net_j)\ \Sigma(\delta_k w_{kj})$
$\partial E/\partial w_{ji} = -output_i\ \delta_j$

j = hidden neuron
i = neuron in previous layer
k = neuron in next layer

Backpropagation

- Calculation of the derivatives flows backwards through the network, hence the name, backpropagation
- These derivatives point in the direction of the maximum increase of the error function
- A small step (learning rate) in the opposite direction will result in the maximum decrease of the (local) error function:

$$w_{new} = w_{old} - \alpha \, \partial E / \partial w_{old}$$

where α is the learning rate

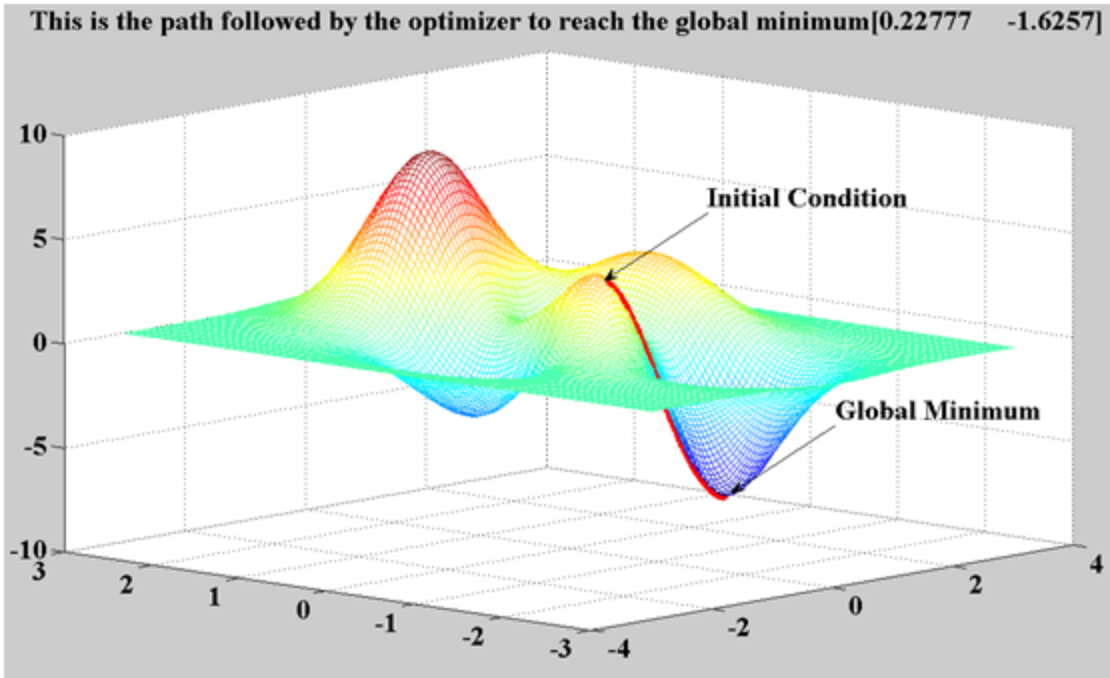
Fundamentals
Classes
Design
Results

Backpropagation

- The learning rate is important
  - Too small
    - Convergence extremely slow
  - Too large
    - May not converge
- Momentum
  - Tends to aid convergence
  - Applies smoothed averaging to the change in weights:

    $\Delta_{new} = \beta\Delta_{old} - \alpha\, \partial E/\partial w_{old}$ β is the momentum coefficient

    $w_{new} = w_{old} + \Delta_{new}$
  - Acts as a low-pass filter by reducing rapid fluctuations


This is the path followed by the optimizer to reach the global minimum[0.22777 -1.6257]
Initial Condition
Global Minimum
10
5
0
-5
-10
3
2
1
0
-1
-2
-3
-4
-2
0
2
4

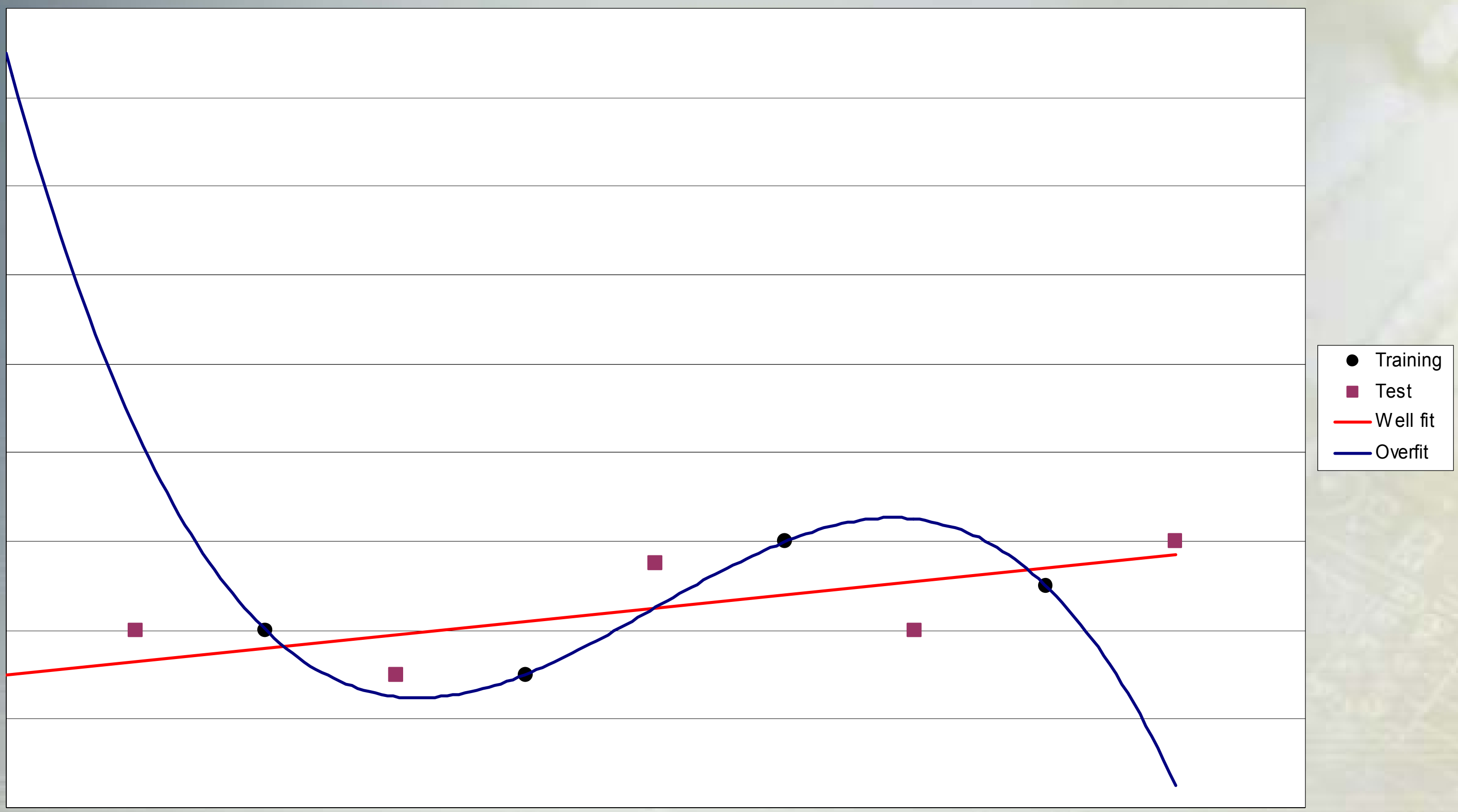
Local Minima

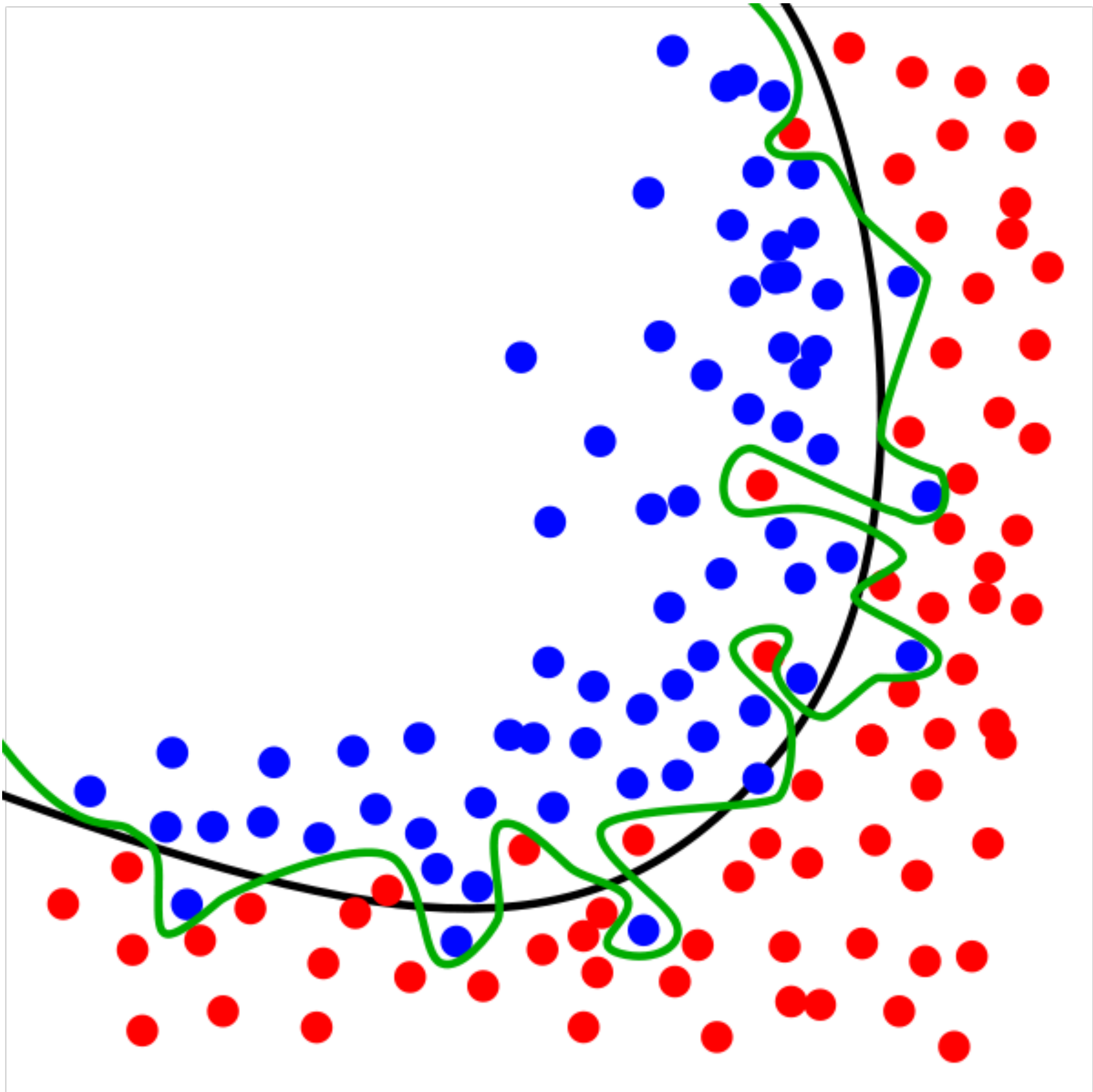
- Training is essentially minimizing the mean square error function
  - Key problem is avoiding local minima
  - Traditional techniques for avoiding local minima:
    - Simulated annealing
      - Perturb the weights in progressively smaller amounts
    - Genetic algorithms
      - Use the weights as chromosomes
      - Apply natural selection, mating, and mutations to these chromosomes

Hidden Layers and Neurons

- For most problems, one layer is sufficient
- Two layers are required when the function is discontinuous
- The number of neurons is very important:
  - Too few
    - Underfit the data – NN can't learn the details
  - Too many
    - Overfit the data – NN learns the insignificant details
  - Start small and increase the number until satisfactory results are obtained

Overfitting





How is the Training Set Chosen?

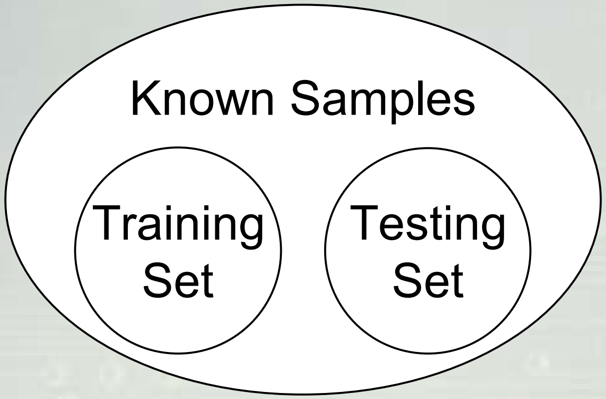
- Overfitting can also occur if a “good” training set is not chosen
- What constitutes a “good” training set?
  - Samples must represent the general population
  - Samples must contain members of each class
  - Samples in each class must contain a wide range of variations or noise effect

Size of the Training Set

- The size of the training set is related to the number of hidden neurons
  - Eg. 10 inputs, 5 hidden neurons, 2 outputs:
  - 11(5) + 6(2) = 67 weights (variables)
  - If only 10 training samples are used to determine these weights, the network will end up being overfit
    - Any solution found will be specific to the 10 training samples
    - Analogous to having 10 equations, 67 unknowns → you can come up with a specific solution, but you can't find the general solution with the given information

Training and Verification

- The set of all known samples is broken into two orthogonal (independent) sets:
  - Training set
    - A group of samples used to train the neural network
  - Testing set
    - A group of samples used to test the performance of the neural network
    - Used to estimate the error rate



Verification

- Provides an unbiased test of the quality of the network
- Common error is to “test” the neural network using the same samples that were used to train the neural network
  - The network was optimized on these samples, and will obviously perform well on them
  - Doesn’t give any indication as to how well the network will be able to classify inputs that weren’t in the training set

Verification

- Various metrics can be used to grade the performance of the neural network based upon the results of the testing set
  - Mean square error, SNR, etc.
- Resampling is an alternative method of estimating error rate of the neural network
  - Basic idea is to iterate the training and testing procedures multiple times
  - Two main techniques are used:
    - Cross-Validation
    - Bootstrapping

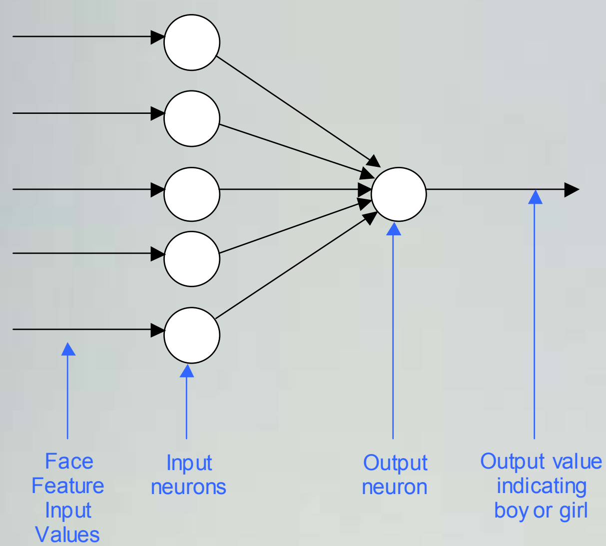
Fundamentals
Classes
Design
Results

Results and Discussion

- A simple toy problem was used to test the operation of a perceptron
- Provided the perceptron with 5 pieces of information about a face – the individual's hair, eye, nose, mouth, and ear type
  - Each piece of information could take a value of +1 or -1
    - +1 indicates a "girl" feature
    - -1 indicates a "guy" feature
- The individual was to be classified as a girl if the face had more "girl" features than "guy" features and a boy otherwise

Results and Discussion

- Constructed a perceptron with 5 inputs and 1 output



- Trained the perceptron with 24 out of the 32 possible inputs over 1000 epochs
- The perceptron was able to classify the faces that were not in the training set

Results and Discussion

- A number of toy problems were tested on multilayer feedforward NN's with a single hidden layer and backpropagation:
  - Inverter
    - The NN was trained to simply output 0.1 when given a "1" and 0.9 when given a "0"
      - A demonstration of the NN's ability to memorize
    - 1 input, 1 hidden neuron, 1 output
    - With learning rate of 0.5 and no momentum, it took about 3,500 epochs for sufficient training
    - Including a momentum coefficient of 0.9 reduced the number of epochs required to about 250

Results and Discussion

- Inverter (continued)
  - Increasing the learning rate decreased the training time without hampering convergence for this simple example
  - Increasing the epoch size, the number of samples per epoch, decreased the number of epochs required and seemed to aid in convergence (reduced fluctuations)
  - Increasing the number of hidden neurons decreased the number of epochs required
    - Allowed the NN to better memorize the training set – the goal of this toy problem
    - Not recommended to use in “real” problems, since the NN loses its ability to generalize

Results and Discussion

- AND gate
  - 2 inputs, 2 hidden neurons, 1 output
  - About 2,500 epochs were required when using momentum
- XOR gate
  - Same as AND gate
- 3-to-8 decoder
  - 3 inputs, 3 hidden neurons, 8 outputs
  - About 5,000 epochs were required when using momentum

Results and Discussion

- Absolute sine function approximator (|sin(x)|)
  - A demonstration of the NN's ability to learn the desired function, |sin(x)|, and to generalize
  - 1 input, 5 hidden neurons, 1 output
  - The NN was trained with samples between $-\pi/2$ and $\pi/2$
    - The inputs were rounded to one decimal place
    - The desired targets were scaled to between 0.1 and 0.9
  - The test data contained samples in between the training samples (i.e. more than 1 decimal place)
    - The outputs were translated back to between 0 and 1
  - About 50,000 epochs required with momentum
  - Not smooth function at 0 (only piece-wise continuous)

Results and Discussion

- Gaussian function approximator (e^{-x^2})
  - 1 input, 2 hidden neurons, 1 output
  - Similar to the absolute sine function approximator, except that the domain was changed to between -3 and 3
  - About 10,000 epochs were required with momentum
  - Smooth function

Results and Discussion

- Primality tester
  - 7 inputs, 8 hidden neurons, 1 output
  - The input to the NN was a binary number
  - The NN was trained to output 0.9 if the number was prime and 0.1 if the number was composite
    - Classification and memorization test
  - The inputs were restricted to between 0 and 100
  - About 50,000 epochs required for the NN to memorize the classifications for the training set
    - No attempts at generalization were made due to the complexity of the pattern of prime numbers
  - Some issues with local minima

Fundamentals
Classes
Design
Results

Results and Discussion

- Prime number generator
  - Provide the network with a seed, and a prime number of the same order should be returned
  - 7 inputs, 4 hidden neurons, 7 outputs
  - Both the input and outputs were binary numbers
  - The network was trained as an autoassociative network
    - Prime numbers from 0 to 100 were presented to the network and it was requested that the network echo the prime numbers
    - The intent was to have the network output the closest prime number when given a composite number
  - After one million epochs, the network was successfully able to produce prime numbers for about 85 - 90% of the numbers between 0 and 100
  - Using Gray code instead of binary did not improve results
  - Perhaps needs a second hidden layer, or implement some heuristics to reduce local minima issues

Conclusion

- The toy examples confirmed the basic operation of neural networks and also demonstrated their ability to learn the desired function and generalize when needed
- The ability of neural networks to learn and generalize in addition to their wide range of applicability makes them very powerful tools

Acknowledgements

- Natural Sciences and Engineering Research Council (NSERC)
- University of Manitoba

References

[AbDo99] H. Abdi, D. Valentin, B. Edelman, *Neural Networks*, Thousand Oaks, CA: SAGE Publication Inc., 1999.

[Hayk94] S. Haykin, *Neural Networks*, New York, NY: Nacmillan College Publishing Company, Inc., 1994.

[Mast93] T. Masters, *Practial Neural Network Recipes in C++*, Toronto, ON: Academic Press, Inc., 1993.

[Scha97] R. Schalkoff, *Artificial Neural Networks*, Toronto, ON: the McGraw-Hill Companies, Inc., 1997.

[WeKu91] S. M. Weiss and C. A. Kulikowski, *Computer Systems That Learn*, San Mateo, CA: Morgan Kaufmann Publishers, Inc., 1991.

[Wass89] P. D. Wasserman, Neural Computing: Theory and Practice, New York, NY: Van Nostrand Reinhold, 1989.