VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Engineering and Computer Science

COMP 307 — Lecture 08

Machine Learning 5

Neural Networks and Back Propagation

Mengjie Zhang (Meng)

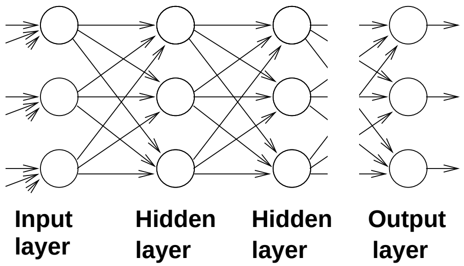
mengjie.zhang@ecs.vuw.ac.nz

Outline

- From perceptron to neural networks
- Network architecture
- Back (error) propagation learning
  - Gradient descent search
  - Relationship between the weight change and the error, outputs
  - Feed forward pass
  - Back propagation pass

Multilayer Perceptron (Neural Networks)

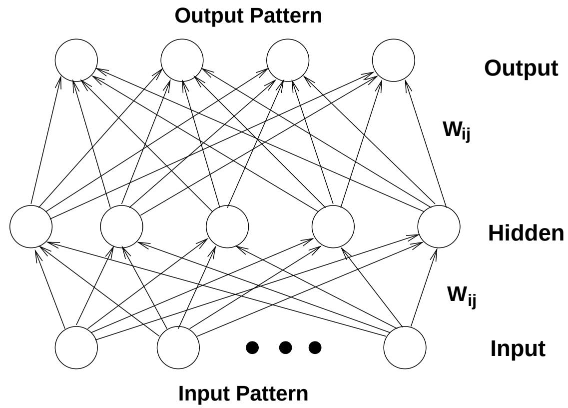
- Change one or two layers of nodes to three or more layers
  - Multilayer perceptron (MLP)
  - Feed forward neural networks
  - Standard feed forward networks: nodes in neighbouring layers are fully connected



  - Input layer: input patterns/features
  - Output layer: output patterns/class labels
  - Hidden layer(s): high level features

Feed Forward Networks as A Pattern Classifier

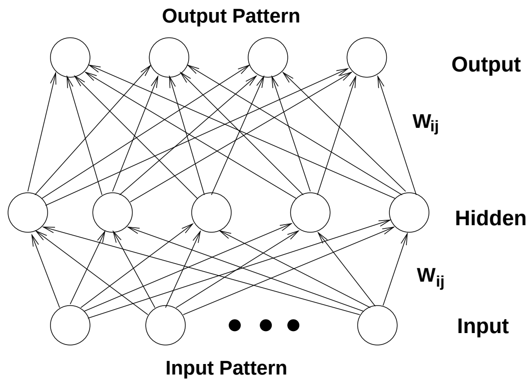
Digits in Postcode	Speech Recognition	Underwater Object	Weather Prediction



Picture of an Envelope	Speech Waveform	Sonar Signal	Weather Data

Networks as A Function Approximator/Series Predictor

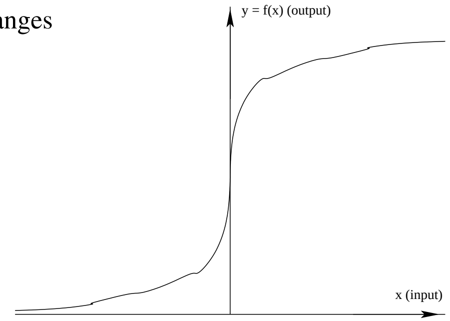
Prediction of activity	Prediction of stock price	Survival months	Genetic Sequence



Sunspot Time Series	Stock market Time Series	Heart attack Data	Genetic Sequences

Changes in Transfer Functions

- Soft threshold:
  - Small changes in weights $\longrightarrow$ small change in output
  - Transfer function, activation function, output function
  - Typically use the sigmoid/logistic function
  - Smooth response to small changes



- The Sigmoid Function:

$$y = f(x) = \frac{1}{1 + e^{-x}}$$

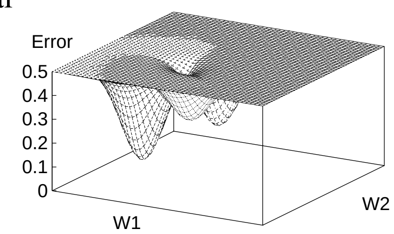
$$y' = \frac{dy}{dx} = f'(x) = f(x)(1 - f(x)) = y(1 - y)$$

Where do Weights Come From

- Massively difficult problem, in general
- Much current research
- General Approach
  1. Get examples for which desired behaviour is known
  2. Pick a random set of weights
  3. Put examples through the network giving network outputs. Difference between network outputs and desired outputs is the error
  4. If error is small enough stop
  5. Adjust current weights to make error smaller
  6. Go to 3

Weight Optimization

- In general: $Error \ = \ f(w_{ij})$
- We want to the minimum value of $Error$
- We have a multidimensional optimization problem



- Need to SEARCH/Learn!!!!

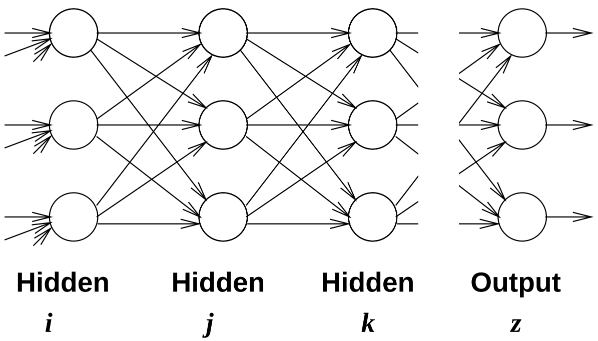
Network Training: Back Error Propagation

- Input Units: Real numbers, usually scaled to be in [0,1]
- Hidden units: Activation $\in [0, 1]$
- Output Units: Activation $\in [0, 1]$
- Usually in fully connected layers, but this is not necessary.
- Transfer function: Logistic
- Units evaluated in serial/synchronous order
- Learning rule: *Generalized delta rule*
- Error of an output unit $d_z - o_z$
- Error of a pattern $\Sigma_z(d_z - o_z)^2$
- Total error of all training patterns
  - Total Sum Squared Error: $TSS = \frac{1}{2}\Sigma_{patterns}\Sigma_z(d_z - o_z)^2$
  - Root Mean Square Error (RMSE) $\sqrt{2TSS/(numpat.numout)}$

Back Propagation: Gradient Descent

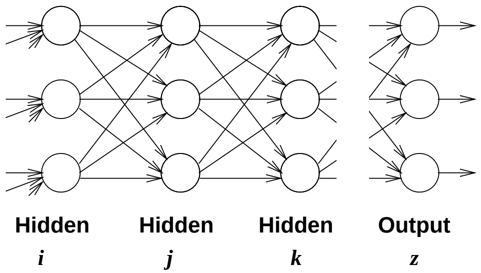
- Hill climbing requires evaluating the effect of one parameter while keeping the others constant
- Gradient descent improvement
  - Requires the ‘hill’ to be a smooth/continuous function of the parameters (weights)
  - Vary all weights simultaneously in proportion to how much good is done by individual changes
  - A move in the direction of the (steepest) gradient
- Back Propagation procedure
  - Relatively efficient procedure for computing how much performance (error reduction) improves with a weight change
  - Computes changes to final layer of weights first
  - Computes changes to next to last layer.....initial layer
  - Requires a smooth transfer function

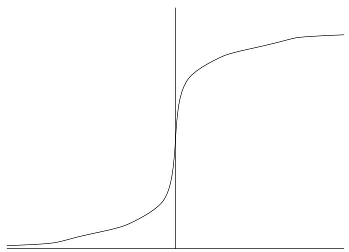
Intuition Behind BP 1



- How big a change should we make to weight $w_{i \to j}$?
- Make a big change if it will result in a big improvement in error
- If a change to $w_{i \to j}$ will have little effect on on error, make it small

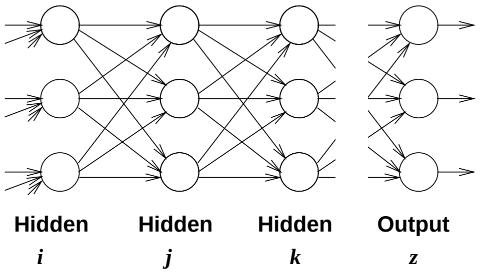
Intuition Behind BP 2





- A change in input to node j results in a change to output that depends on the *slope* of transfer function
- Change in input has maximum effect where the slope is steepest
- Slope of sigmoid/logistic is given by $o(1 - o)$
  Thus $\Delta w_{i \to j} \propto o_j(1 - o_j)$

Intuition Behind BP 3

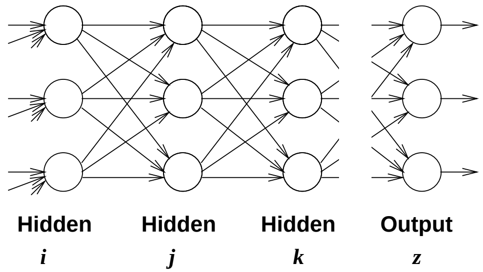


- Change in input to node j depends on output of node i. $w_{i \to j}$ should change substantially if o_i is high. Thus $\Delta w_{i \to j} \propto o_i$
- Let β be a factor which measures how beneficial the change is (in terms of lower error). Node j is connected to nodes in next (kth) layer. A change in o_j will be a benefit to each one. So

  Hidden: $\beta_j = \Sigma_k w_{j \to k} o_k (1 - o_k) \beta_k$

  Output: $\beta_z = d_z - o_z$ and $\boxed{\mathbf{\Delta w_{i \to j} \propto \beta_j}}$

Intuition Behind BP 4



- Putting all together:

  $\Delta w_{i \to j} \propto o_i o_j (1 - o_j) \beta_j$

  Let η be the constant or *learning rate*
- **Back-propagation formulas**

  $\Delta w_{i \to j} = \eta o_i o_j (1 - o_j) \beta_j$

  $\beta_j = \Sigma_k w_{j \to k} o_k (1 - o_k) \beta_k$ (Hidden units)

  $\beta_z = d_z - o_z$ (Output Units)

BP Algorithm

- Let η be the learning rate.
- Set all weights, including biases to small random values.
- Until total error (TSS or RMSE) is small enough do
  - For each input vector
    - Feed forward pass to get outputs
    - Compute β for output nodes $\beta_z = d_z - o_z$
    - Compute β for hidden nodes, working from last layer to first layer $\beta_j = \Sigma_k w_{j \to k} o_k (1 - o_k) \beta_k$
    - Compute and store weight changes for all weights $\Delta w_{i \to j} = \eta o_i o_j (1 - o_j) \beta_j$
  - Add up weight changes for all input vectors and change the weights

Summary

- Multilayer perceptron vs feed forward networks
- Network architecture
- NN applications
- Back (error) propagation learning
  - Gradient descent search
  - Relationship between the weight change and the error, outputs
  - Feed forward pass
  - Back propagation pass