Multi-Layer Neural Networks
[RN2] Sec 20.5
[RN3] Sec 20.5

CS 486/686

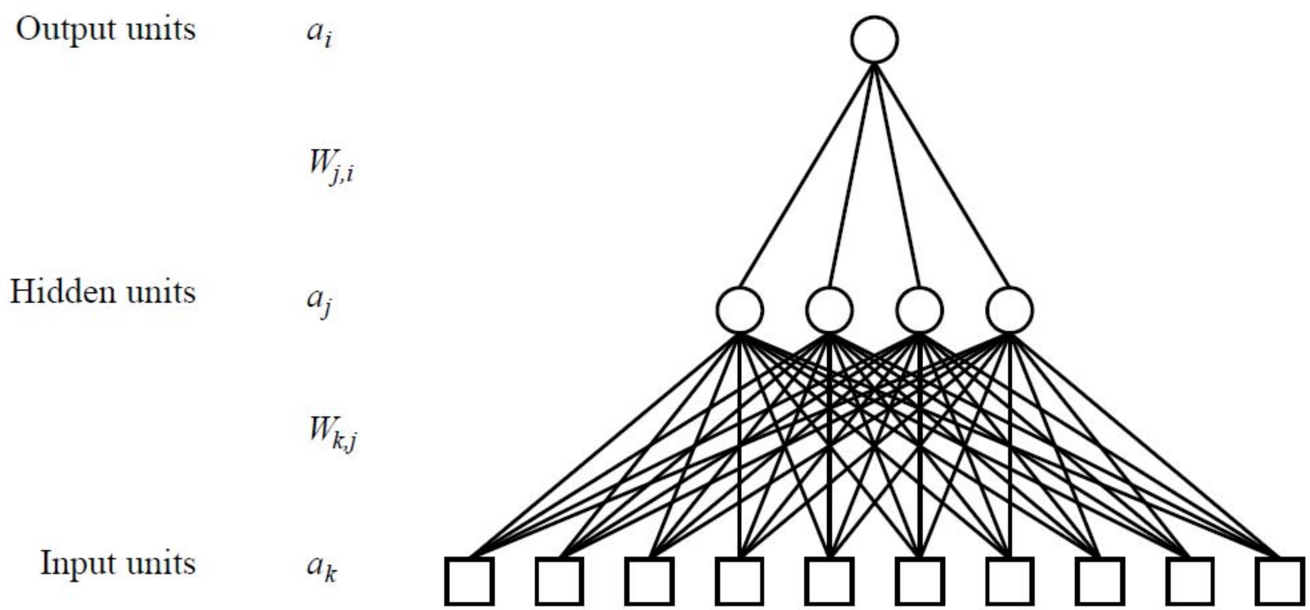
University of Waterloo

Lecture 20: July 9, 2015

Multilayer Feed-forward Neural Networks

- Perceptron can only represent (soft) linear separators
  - Because single layer
- With multiple layers, what fns can be represented?
  - Virtually any function!

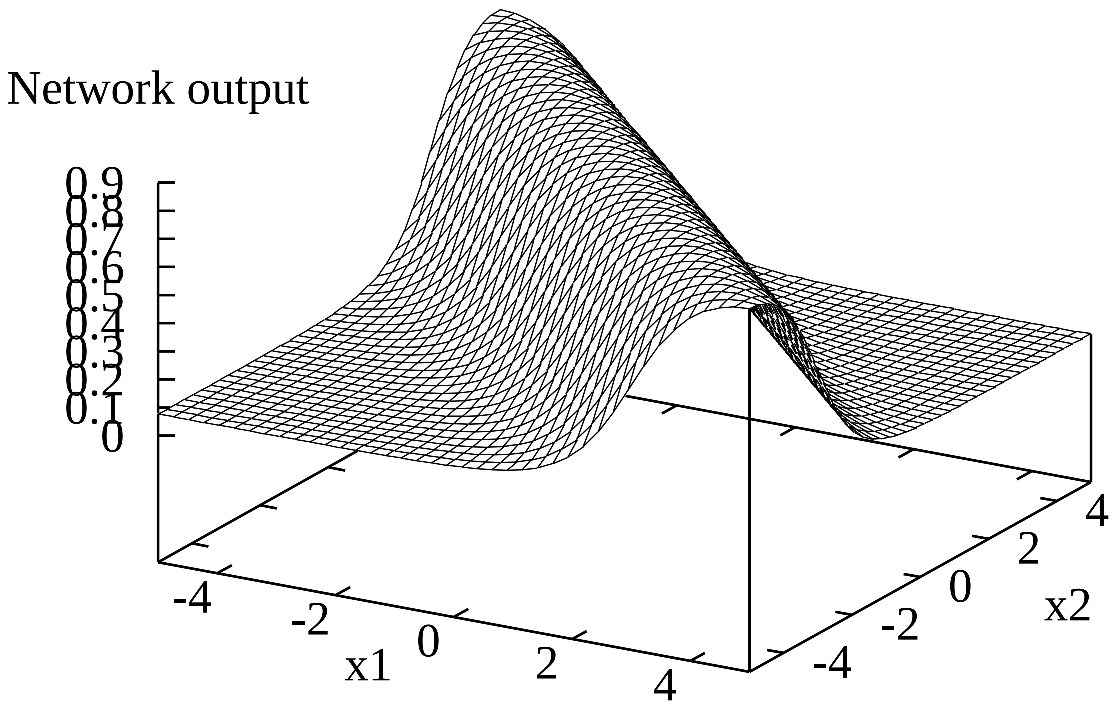
Multilayer Networks



$$a_i = g\left(\sum_j W_{ji}\, g\left(\sum_k W_{kj} a_k\right)\right)$$

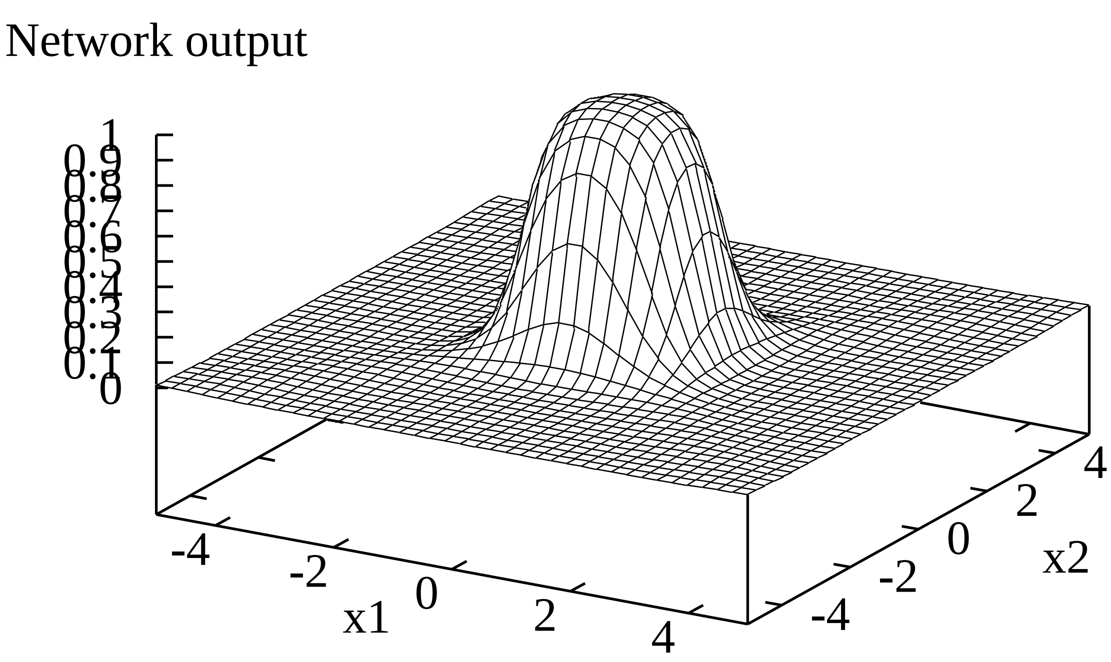
Multilayer Networks

- Adding two sigmoid units with parallel but opposite "cliffs" produces a ridge



Multilayer Networks

- Adding two intersecting ridges (and thresholding) produces a bump



Multilayer Networks

- By tiling bumps of various heights together, we can approximate any function
- **Theorem:** Neural networks with at least one hidden layer of sufficiently many sigmoid units can approximate any function arbitrarily closely.

Common Activation Functions

- Threshold: $h(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$
- Sigmoid: $h(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
- Gaussian: $h(x) = e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$
- Hyperbolic tangent: $h(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Identity: $h(x) = x$

Weight Training

- Parameters: $< \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots >$
- Objectives:
  - **Error minimization**
    - **Backpropagation (aka "backprop")**
  - Maximum likelihood
  - Maximum a posteriori
  - Bayesian learning

Least squared error

- Error function

$$E(\boldsymbol{W}) = \frac{1}{2}\sum_n E_n(\boldsymbol{W})^2 = \frac{1}{2}\sum_n \left|\left|f(\boldsymbol{x_n}, \boldsymbol{W}) - y_n\right|\right|_2^2$$

where $\boldsymbol{x_n}$ is the input of the n^{th} example
y_n is the label of the n^{th} example
$f(\boldsymbol{x_n}, \boldsymbol{W})$ is the output of the neural net

Sequential Gradient Descent

- For each example $(\boldsymbol{x}_n, y_n)$ adjust the weights as follows:

$$W_{ji} \leftarrow W_{ji} - \alpha \frac{\partial E_n}{\partial W_{ji}}$$

- How can we compute the gradient efficiently given an arbitrary network structure?
- Answer: **backpropagation algorithm**

Backpropagation

- Back-Prop-Learning(examples,network)
  - Repeat
    - For each example e do
      - Compute output a of each node in **forward** pass
        - » Input nodes: $a_j \leftarrow x_j[e]$
        - » Other nodes: $in_i \leftarrow \sum_j W_{ji} a_j$ and $a_i \leftarrow g(in_i)$
      - Compute modified error Δ of each node in **backward** pass ($l = L$ to 1)
        - » Output nodes: $\Delta_i \leftarrow g'(in_i)\,(y_i[e] - a_i)$
        - » For each node j in layer l: $\Delta_j \leftarrow g'\left(in_j\right) \sum_i W_{ji} \Delta_i$
          - » For each node i in layer $l+1$: $W_{ji} \leftarrow W_{ji} + \alpha\, a_j\, \Delta_i$
  - Until some stopping criteria satisfied
  - Return learnt network

Forward phase

- Propagate inputs forward to compute the output of each unit
- Output a_i at unit i:

$$a_i = g(in_i) \quad \text{where} \quad in_i = \sum_j W_{ji} a_j$$

Backward phase

- Use chain rule to recursively compute gradient
  - For each weight W_{ji}: $\frac{\partial E_n}{\partial W_{ji}} = \frac{\partial E_n}{\partial in_i}\frac{\partial in_i}{\partial W_{ji}} = \Delta_i a_j$
  - Let $\Delta_i \equiv \frac{\partial E_n}{\partial in_i}$ then

$$\Delta_i = \begin{cases} g'(in_i)(y_i - a_i) & \text{base case: } i \text{ is an output unit} \\ g'(in_i)\sum_j W_{ji}\Delta_j & \text{recursion: } i \text{ is a hidden unit} \end{cases}$$

  - Since $in_i = \sum_j W_{ji} a_j$ then $\frac{\partial in_i}{\partial W_{ji}} = a_j$

Simple Example

- Consider a network with two layers:
  - Hidden nodes: $g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
    - Tip: $tanh'(x) = 1 - tanh^2(x)$
  - Output node: $g(x) = x$

- Objective: squared error

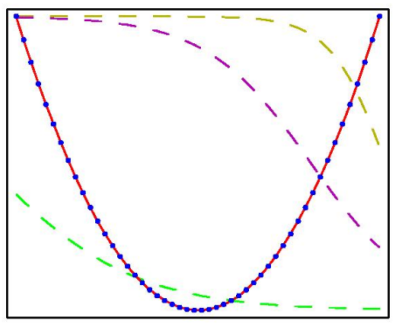
Simple Example

- Forward propagation:
  - Hidden units: $in_j =$ $a_j =$
  - Output units: $in_i =$ $a_i =$
- Backward propagation:
  - Output units: $\Delta_i =$
  - Hidden units: $\Delta_j =$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial W_{kj}} =$
  - Output layer: $\frac{\partial E_n}{\partial W_{ji}} =$

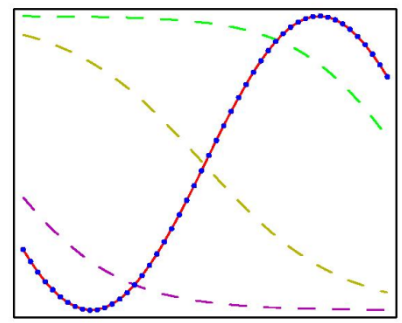
Non-linear regression examples

- Two layer network:
  - 3 tanh hidden units and 1 identity output unit

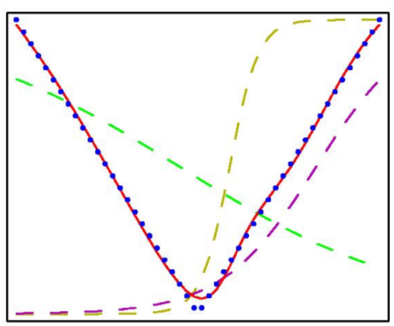
$y = x^2$



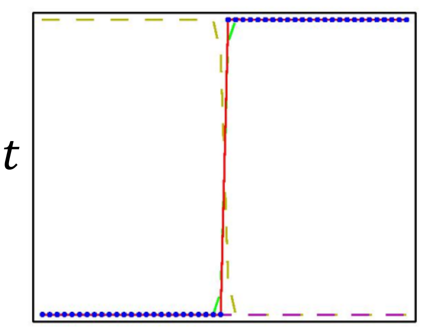
$y = \sin x$



$y = |x|$



$y = \int_{-\infty}^{x} \delta(t)dt$



Analysis

- Efficiency:
  - Fast gradient computation: linear in number of weights
- Convergence:
  - Slow convergence (linear rate)
  - May get trapped in local optima
- Prone to overfitting
  - Solutions: early stopping, regularization (add $||w||_2^2$ penalty term to objective)

Neural Net Applications

- Neural nets can approximate any function, hence 1000's of applications
  - Speech recognition
  - Character recognition
  - Paint-quality inspection
  - Vision-based autonomous driving
  - Etc.