CS786
Lecture 20: July 10th, 2012

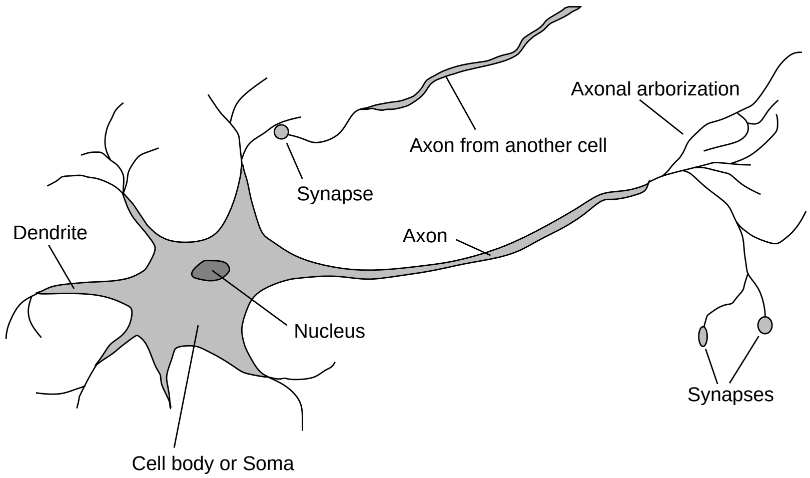
Neural Networks

[Bishop, Pattern Recognition and Machine Learning] Sections 5.2, 5.3

Outline

- Neural networks
- Sigmoid belief networks
  - Feed-forward neural networks with sigmoid activation function
  - Bayesian network
- Deep belief networks
  - Recurrent neural networks with sigmoid activation function
  - Markov networks

Neuron



Artificial Neural Networks

- Idea: **mimic the brain to do computation**
- Artificial neural network:
  - Nodes (a.k.a. units) correspond to neurons
  - Links correspond to synapses
- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal corresponds to neurons firing rate

ANN Unit

- For each unit j:
- **Weights: $\boldsymbol{W}$**
  - Strength of the link from unit i to unit j
  - Input signals x_i weighted by W_{ji} and linearly combined: $a_j = \sum_i W_{ji}\, x_i + w_0 = \boldsymbol{W_j^T}\overline{\boldsymbol{x}}$
- **Activation function: $\boldsymbol{h}$**
  - Numerical signal produced: $y_j = h(a_j)$

ANN Unit

- Picture

Activation Function

- Should be nonlinear
  - Otherwise network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be “active” (output near 1) when fed with the “right” inputs
  - Unit should be “inactive” (output near 0) when fed with the “wrong” inputs

Common Activation Functions

Threshold

Sigmoid

Other activation functions: Identity: $h(a) = a$

Gaussian: $h(a) = e^{-\frac{1}{2}\left(\frac{a-\mu}{\sigma}\right)^2}$ Tanh: $h(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$

Network Structures

- **Feed-forward network**
  - Directed **acyclic** graph
  - No internal state
  - Simply computes outputs from inputs
- **Recurrent network**
  - Directed **cyclic** graph
  - Dynamical system with internal states
  - Can memorize information

Two-Layer Architecture

- Feed-forward neural network

- Hidden units: $z_j = h_1(\boldsymbol{w}_j^{(1)} \overline{\boldsymbol{x}})$
- Output units: $y_k = h_2(\boldsymbol{w}_k^{(2)} \overline{\boldsymbol{z}})$
- Overall: $y_k = h_2\left(\sum_j w_{kj}^{(2)} h_1\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$

Weight training

- Parameters: $< \boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \ldots >$
  - Where $\boldsymbol{W}^{(i)}$ is the matrix of weights in layer i
- 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 n is an index denoting each data point $(\boldsymbol{x_n}, y_n)$

- $f(\boldsymbol{x}, \boldsymbol{W})$ is the function computed by the neural network
  - E.g., two layer network with sigmoid activation functions

$$f(\boldsymbol{x}, \boldsymbol{W}) = \sigma\left(\sum_j w_{kj}^{(2)} \sigma\left(\sum_i w_{ji}^{(1)} x_i\right)\right)$$

Sequential Gradient Descent

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

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_n}{\partial w_{ji}}$$

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

Backpropagation Algorithm

- Two phases:
  - Forward phase: compute output z_j of each unit j
  - Backward phase: compute delta δ_j at each unit j

Forward phase

- Propagate inputs forward to compute the output of each unit
- Output z_j at unit j:

  $z_j = h(a_j)$ where $a_j = \sum_i w_{ji} z_i$

Backward phase

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

    $$\delta_j = \begin{cases} h'(a_j)(y_j - z_j) & \text{base case: } j \text{ is an output unit} \\ h'(a_j) \sum_k w_{kj} \delta_k & \text{recursion: } j \text{ is a hidden unit} \end{cases}$$

  - Since $a_j = \sum_i w_{ji} z_i$ then $\frac{\partial a_j}{\partial w_{ji}} = z_i$

Simple Example

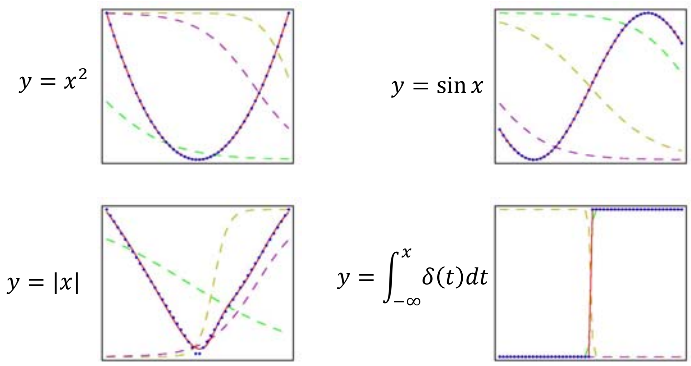
- Consider a network with two layers:
  - Hidden nodes: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
    - Tip: $tanh'(a) = 1 - (tanh(a))^2$
  - Output node: $h(a) = a$
- Objective: squared error

Simple Example

- Forward propagation:
  - Hidden units: $a_j =$ $z_j =$
  - Output units: $a_k =$ $z_k =$
- Backward propagation:
  - Output units: $\delta_k =$
  - Hidden units: $\delta_j =$
- Gradients:
  - Hidden layers: $\frac{\partial E_n}{\partial w_{ji}} =$
  - Output layer: $\frac{\partial E_n}{\partial w_{kj}} =$

Non-linear regression examples

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



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)