Multi-layer Neural Networks,
Error Backpropagation
[D] Chapt. 10, [HTF] Chapt. 11, [B] Sec. 5.2, 5.3, [M] Sec. 16.5, [RN] Sec. 18.7
Quick Recap: Linear Models

Linear Regression                  Linear Classification
Quick Recap: Non-linear Models

Non-linear classification        Non-linear regression
Non-linear Models

• Convenient modeling assumption: linearity
• Extension: non-linearity can be obtained by mapping $x$ to a non-linear feature space $\phi(x)$
• Limit: the basis functions $\phi_i(x)$ are chosen a priori and are fixed

• Question: can we work with unrestricted non-linear models?
Flexible Non-Linear Models

- Idea 1: Select basis functions that correspond to the training data and retain only a subset of them (e.g., **Support Vector Machines**)

- Idea 2: Learn non-linear basis functions (e.g., **Multi-layer Neural Networks**)

CS489/698 (c) 2018 P. Poupart
Two-Layer Architecture

• Feed-forward neural network

• Hidden units: $z_j = h_1(w^{(1)}_j x)$

• Output units: $y_k = h_2(w^{(2)}_k z)$

• Overall: $y_k = h_2 \left( \sum_j w^{(2)}_{kj} h_1 \left( \sum_i w^{(1)}_{ji} x_i \right) \right)$
Common activation functions $h$

- **Threshold**: $h(a) = \begin{cases} 
1 & a \geq 0 \\
-1 & a < 0 
\end{cases}$

- **Sigmoid**: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$

- **Gaussian**: $h(a) = e^{-\frac{1}{2}(\frac{a-\mu}{\sigma})^2}$

- **Tanh**: $h(a) = \tanh(a) = \frac{e^a-e^{-a}}{e^a+e^{-a}}$

- **Identity**: $h(a) = a$
Adaptive non-linear basis functions

• Non-linear regression
  – \( h_1 \): non-linear function and \( h_2 \): identity

• Non-linear classification
  – \( h_2 \): non-linear function and \( h_2 \): sigmoid
Weight training

• Parameters: $< \mathbf{W}^{(1)}, \mathbf{W}^{(2)}, ... >$

• Objectives:
  – Error minimization
    • Backpropagation (aka “backprop”)
  – Maximum likelihood
  – Maximum a posteriori
  – Bayesian learning
Least squared error

- Error function

\[ E(W) = \frac{1}{2} \sum_n E_n(W)^2 = \frac{1}{2} \sum_n \| f(x_n, W) - y_n \|_2^2 \]

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

then we are optimizing a linear combination of non-linear basis functions
Sequential Gradient Descent

• For each example \((x_n, y_n)\) 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 $\delta_j$ at each unit $j$
Forward phase

• Propagate inputs forward to compute the output of each unit

• Output $z_j$ at unit $j$:

$$z_j = h(a_j) \quad \text{where} \quad a_j = \sum_i w_{ji} z_i$$
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 $\delta_j \equiv \frac{\partial E_n}{\partial a_j}$ then

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

- 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: \( a_j = \)
  – Output units: \( a_k = \)

• 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

\[ y = x^2 \]

\[ y = \sin x \]

\[ y = |x| \]

\[ y = \int_{-\infty}^{x} \delta(t)dt \]
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), dropout