Lecture 13: Neural Networks
CS486/686 Intro to Artificial Intelligence

Pascal Poupart
David R. Cheriton School of Computer Science
Outline

- Neural networks
  - Perceptron
  - Supervised learning algorithms for neural networks
Neuron

Dendrite

Synapse

Axon from another cell

Axonal arborization

Nucleus

Cell body or Soma

Synapses
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 $i$:

- **Weights:** $W$
  - Strength of the link from unit $i$ to unit $j$
  - Input signals $x_i$ weighted by $W_{ji}$ and linearly combined:
    \[
    a_j = \sum_i W_{ji} x_i + W_{j0} = W_j \bar{x}
    \]

- **Activation function:** $h$
  - Numerical signal produced: $y_j = h(a_j)$
ANN Unit

- Picture

\[ w_{j0}, w_{j1}, w_{j2}, w_{j3} \]

\[ \sum x_i w_i = a_j \]

\[ h(a_j) = \text{out}_j \]
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
Logic Gates

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean functions

- What should be the weights of the following units to code AND, OR, NOT?

\[
\begin{align*}
\alpha &= w_0 + w_1 x_1 + w_2 x_2 \\
\sigma(a) &= \begin{cases} 
0 & \text{if } a \leq 0 \\
1 & \text{otherwise}
\end{cases}
\end{align*}
\]
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
Feed-forward network

- Simple network with two inputs, one hidden layer of two units, one output unit
Perceptron

- Single layer feed-forward network
Threshold Perceptron Hypothesis Space

- Hypothesis space $h_w$:
  - All binary classifications with parameters $w$ s.t.
    
    \[
    w^T \bar{x} > 0 \rightarrow +1 \\
    w^T \bar{x} < 0 \rightarrow -1
    \]

- Since $w^T \bar{x}$ is linear in $w$, perceptron is called a \textit{linear separator}
Linear Separability

- Are all Boolean gates linearly separable?

(a) $I_1 \text{ and } I_2$

(b) $I_1 \text{ or } I_2$

(c) $I_1 \text{ xor } I_2$
Sigmoid Perceptron

- Represent “soft” linear separators
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 $h$

- **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**: $h(a) = a$
Weight training

- Parameters: $< W^{(1)}, W^{(2)}, ... >$

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

- Error function

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

where \( x_n \) is the input of the \( n^{th} \) example

\( y_n \) is the label of the \( n^{th} \) example

\( f(x_n, W) \) is the output of the neural net
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

- Today: automatic differentiation
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 $w_{ji}$: 
    \[
    \frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i
    \]

- 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: \( a_j = \sum_i w_{ji} x_i \) \( z_j = \tanh (a_j) \)
  - Output units: \( a_k = \sum_j w_{kj} z_j \) \( z_k = a_k \)

- **Backward propagation:**
  - Output units: \( \delta_k = z_k - y_k \)
  - Hidden units: \( \delta_j = (1 - z_j^2) \sum_k w_{kj} \delta_k \)

- **Gradients:**
  - Hidden layers: \( \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i = (1 - z_j^2) \sum_k w_{kj} \delta_k x_i \)
  - Output layer: \( \frac{\partial E_n}{\partial w_{kj}} = z_k \delta_j = (z_k - y_k) z_j \)
Non-linear regression examples

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

\[
y = x^2 \quad y = \sin x
\]

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