Neural Networks

Outline

- Perceptron: limitations;
- Feedforward networks and Backpropagation:
What is a Neural Network, anyway?

- Often associated with biological devices (brains), electronic devices, or network diagrams;
- But the best conceptualization for this presentation is **none** of these: **think of a neural network as a mathematical function**

The pros of Neural Networks

- Successfully used on a variety of domains:
  
  - PC games, Business strategy, Buyer prospect selection, Stock market analysis, Consumer price forecasts, Cost analysis, Employee selection, Intelligent software applications, Legal strategies, Managerial decision making, Personnel profiling, Process control, Quality control, Real estate market forecasting, Sales forecasts, Security analysis, Spectral analysis, Stock market analysis, Temperature and weather prediction, Troubleshooting and much more.

- Can provide solutions to very complex and nonlinear problems:
- If provided with sufficient amount of data, can solve classification and forecasting problems accurately and easily
- Once trained, prediction is fast;
Introduction: A Simple Architecture

\[
g(x) = \sum_{j=1}^{n} w_j x_j
\]

Representational Power of Perceptrons

- Marvin Minsky and Seymour Papert, "Perceptron" 1969:
  "The perceptron can solve only problems with linearly separable classes."
- Examples of linearly separable Boolean functions:

AND

OR
Representational Power of Perceptrons

Perceptron that computes the AND function

Perceptron that computes the OR function

Example of a non linearly separable Boolean function:

The EX-OR function cannot be computed by a perceptron
Adding a Hidden Layer

Multilayer Neural Networks

- Generalization of a perceptron;
- Achieve increased computational power;
- Idea: Introduce layers of units between the input and the output units:
Multilayer Neural Networks

- Allow to learn non-linearly separable transformations from input to output;
- A single hidden layer allows to compute any input/output transformation;

Example: EX-OR

- Consider first a perceptron:

- Correct answer in three cases:
  - \(x_1 = 0, x_2 = 0\) \(H(-0.5) = 0\)
  - \(x_1 = 1, x_2 = 0\) \(H(1-0.5) = 1\)
  - \(x_1 = 0, x_2 = 1\) \(H(1-0.5) = 1\)
- 4th case:
  - \(x_1 = 1, x_2 = 1\) \(H(1+1-0.5) = 1\) Wrong!
Example: EX-OR (contd.)

- **Idea**: Introduce one hidden unit with a large enough threshold, so that it is activated only in the 4th case. The hidden unit provides a negative input to the output unit to correct its response in the 4th case.

  
  - First three cases: as before. **OK**
  - 4th case: \( x_1 = 1, x_2 = 1\)
    \[
    H(1 + 1 - 2 - 0.5) = 0 \quad \text{OK!}
    \]

---

**Multilayer Neural Networks**

- **Activation function**:
  Differentiable function \( g \):
  
  \[
  y = g\left( \sum_k w_k x_k \right) \in (0, 1)
  \]

- **Network’s dynamic**:
  - \( f \): target transformation (unknown) from input to output;
  - For each configuration \( x \) of the input layer, the network computes a configuration \( y \) of the output layer;
  - The network adjusts the weights so that, after a finite number of steps, the network’s output \( y \sim f(x) \)

- **Criterion**:
  - Minimize the difference between the network’s response and the desired output.
Learning Algorithm: No Hidden Units first

\[ z = w^T x \]
\[ y = g(z) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \]

Sigmoid function

\[ J(w) = \frac{1}{2} (y^* - y)^2 \]
\[ y = g(w) = \frac{1}{1 + e^{-w^T x}} \]
\[ \frac{\partial y}{\partial w_i} = y(1 - y)x_i \]
\[ \frac{\partial J}{\partial w_i} = -(y^* - y)(1 - y)x_i \]

By applying gradient descent:
\[ \Delta w_i = \mu y(1 - y)(y^* - y)x_i \]
\[ \Delta w_i = w_{i+1} - w_i \]

\( \mu \) is the learning rate
Generalization of Delta Rule for Feedforward Networks

Fixed target function we want to learn: \( t_k = f(x_k) \)

Error over input \( x_k \):
\[
E_k = \frac{1}{2} \sum_j (t_{kj} - y_{kj})^2
\]

Total error:
\[
E = \sum_k E_k
\]

Backpropagation algorithm: provides an efficient procedure to compute derivatives

Backpropagation algorithm:
\[
z_{kj} = \sum_i w_{ji} o_{ki}
\]

\[
o_{kj} = g_j(z_{kj}), \quad g_j \text{ differentiable in } z_{kj}
\]

Goal: learn the weights so that the mean squared error is minimized
**Backpropagation**

Fixed target function we want to learn: \( t_k = f(x_k) \)

Error over input \( x_k \):
\[
E_k = \frac{1}{2} \sum_j (t_{kj} - y_{kj})^2
\]

Total error:
\[
E = \sum_k E_k
\]

We want:
\[
\Delta_k w_{ji} \propto -\frac{\partial E_k}{\partial w_{ji}}
\]

\[
\frac{\partial E_k}{\partial w_{ji}} = \frac{\partial E_k}{\partial z_{ij}} \frac{\partial z_{ij}}{\partial w_{ji}}
\]

\[
z_{kj} = \sum_i w_{ji} o_{ki}
\]

Let's define:
\[
\frac{\partial E_k}{\partial z_{ij}} = -\delta_{ij} \Rightarrow \frac{\partial E_k}{\partial w_{ji}} = -\delta_{kj} o_{ki}
\]

Thus: to perform a gradient descent on the surface of \( E \) we need to modify the weights as:
\[
\Delta_k w_{ji} = \mu \delta_{kj} o_{ki}
\]

---

**Backpropagation**

We need to compute the values \( \delta_j \):
\[
\delta_j = -\frac{\partial E_k}{\partial z_{ij}} = -\frac{\partial E_k}{\partial o_{kj}} \frac{\partial o_{kj}}{\partial z_{ij}}
\]

From:
\[
o_{kj} = g_j(z_{kj}) \quad \frac{\partial o_{kj}}{\partial z_{ij}} = g'_j(z_{kj})
\]

To compute \( \frac{\partial E_k}{\partial v_{ij}} \) we distinguish two cases:

- 1st case: \( u_j \) is an output unit
- 2nd case: \( u_j \) is a hidden unit
Backpropagation

1\textsuperscript{st} case: \( u_j \) is a output unit

\[ b e c a u s e \ E_k = \frac{1}{2} \sum_j (t_{kj} - y_{kj})^2 \]

\[ \frac{\partial E_k}{\partial o_{kj}} = -(t_{kj} - y_{kj}) = -(t_{kj} - o_{kj}) \]

\[ \Rightarrow \delta_{kj} = (t_{kj} - o_{kj})g_j'(z_{kj}) \]

2\textsuperscript{nd} case: \( u_j \) is a hidden unit

\[ \frac{\partial E_k}{\partial o_{kj}} = \sum_t \frac{\partial E_k}{\partial z_{kt}} \frac{\partial z_{kt}}{\partial o_{kj}} = \sum_t \frac{\partial E_k}{\partial o_{kj}} \frac{\partial}{\partial o_{kj}} \left( \sum_l w_{lj} o_{kl} \right) = \]

\[ \sum_t \frac{\partial E_k}{\partial z_{kt}} w_{lj} = -\sum_t \delta_{kl} w_{lj} \]

\[ \Rightarrow \delta_{kj} = g_j'(z_{kj}) \sum_t \delta_{kl} w_{lj} \]

Recursive procedure to compute \( \delta \) for all the units of the network!

\[ \Delta_k w_{ji} = \mu \delta_{kj} o_{ki} \]

Such \( \delta \) are used in:
Wrapping up the Backpropagation Algorithm

Three key equations:

- **Generalized Delta rule:**
  \[ \Delta_k w_{ji} = \mu \delta_{kj} o_{ki} \]

- **For output units, the error signal is:**
  \[ \delta_{kj} = (t_{kj} - o_{kj}) g_j'(z_{kj}) \]

- **For hidden units, the error signal is:**
  \[ \delta_{kj} = g_j'(z_{kj}) \sum_t \delta_{kt} w_{ij} \]

Backpropagation: Summary

- **Activation:** each input unit \( u_j \) is given the state \( x_{ij} \)
- **Signal propagation:** For each hidden and output unit, compute
  \[ o_{kj} = g_j \left( \sum_i w_{ji} o_{ki} \right) \]
- **Comparison:** For each output unit \( u_j \), compute:
  \[ \delta_{kj} = (t_{kj} - o_{kj}) g_j'(\sum_i w_{ji} o_{ki}) \]
- **Backpropagation:** (the computed \( \delta \) become the input of the reversed network) For each hidden unit \( u_j \), compute:
  \[ \delta_{ki} = f_j' \left( \sum_i w_{ji} o_{ki} \right) \sum_t \delta_{kt} w_{ij}^{k-1} \]
- **Weight Update:**
  \[ w_{ji}^k = w_{ji}^{k-1} + \mu \delta_{kj} o_{ki} \]
Learning Rate and Momentum

- \( \mu \) too small \( \Rightarrow \) very slow learning rate
- \( \mu \) too large \( \Rightarrow \) oscillating behavior
- We want to set \( \mu \) as large as possible avoiding oscillations
- Solution: introduce momentum in the learning rule. The momentum includes the direction of the previous update:

\[
\Delta w_{ji}(n+1) = \mu \delta_{kj} o_{ki} + \alpha \Delta w_{ji}(n)
\]

\( \alpha = 0.9 \)

Backpropagation: applications

- Perhaps the most successful and widely used learning algorithm for NNs;
- Used in a variety of domains:
  - clinical diagnosis,
  - predicting protein structure,
  - character recognition,
  - fingerprint recognition,
  - modeling residual chlorine decay in water,
  - weather forecast,
  - waveform recognition,
  - backgammon, etc.
References

- Original paper on backpropagation: