Backpropagation Through Time: What It Does and How to Do It

Paul J. Werbos

Presenter: Tina Liu, 2015/09/08
Recap: Supervised Learning Problem

- Example: Hand-written digit recognition
- 2000 examples of handwritten digits:
  - $T = 2000$
- Each input is a 19*20 image
  - $X(t) = [X_1(t), X_2(t), ..., X_m(t)], m = 380, 1 \leq t \leq 2000$
- The answer is a digit: $Y(t) = [Y_n(t)], n = 1$
- The output of the network: $\hat{Y}(t)$
To solve this, we need to specify

- The topology: how the network connects
- A learning rule: how to find the best weights $\mathbf{W}$ so that $\hat{Y}(t)$ approximate $Y(t)$

Backpropagation!
Feedforward Network

• Perceptron
Feedforward Network

- Perceptron

\[ x_1 W_{11} + x_2 W_{12} + \cdots + x_m W_{13} = net_1 \]

\[ x_{m+1}(Y_{m+1-N}) \]
Feedforward Network

- Perceptron

$$x_i = X_i, \quad 1 \leq i \leq m$$

$$\text{net}_i = \sum_{j=1}^{i-1} W_{ij} x_i, \quad m < i \leq N + n$$

$$x_i = s(\text{net}_i), \quad m < i \leq N + n$$

$$Y_i = x_{i+N}, \quad 1 \leq i \leq n$$

$$s(z) = \frac{1}{1 + e^{-z}}$$
Now comes backpropagation
Now comes backpropagation

\[
E = \sum_{t=1}^{T} E(t) = \sum_{t=1}^{T} \sum_{i=1}^{n} \left( \frac{1}{2} (\hat{Y}_i(t) - Y_i(t))^2 \right).
\]

Find

\[ W = \text{argmin } E \]
\[ x_i = X_i, \quad 1 \leq i \leq m \]
\[ \text{net}_i = \sum_{j=1}^{i-1} W_{ij} x_j, \quad m < i \leq N + n \]
\[ x_i = s(\text{net}_i), \quad m < i \leq N + n \]
\[ Y_i = x_{i+N}, \quad 1 \leq i \leq n \]
\[ x_i = x_i, \quad 1 \leq i \leq m \]
\[ \text{net}_i = \sum_{j=1}^{i-1} W_{ij} x_i, \quad m < i \leq N + n \]
\[ x_i = s(\text{net}_i), \quad m < i \leq N + n \]
\[ Y_i = x_{i+N}, \quad 1 \leq i \leq n \]

\[
F \_ W_{ij} = \frac{\partial E}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} * x_j
\]

\[ = \sum_{t=1}^{T} F \_ \text{net}_i * x_j \]
\[ x_i = x_i, \quad 1 \leq i \leq m \]

\[ \text{net}_i = \sum_{j=1}^{i-1} W_{ij} x_j, \quad m < i \leq N + n \]

\[ x_i = s(\text{net}_i), \quad m < i \leq N + n \]

\[ Y_i = x_{i+N}, \quad 1 \leq i \leq n \]

\[ F_{-W_{ij}} = \frac{\partial E}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} \ast x_j \]

\[ = \sum_{t=1}^{T} F_{-\text{net}_i} \ast x_j \]

\[ F_{-\text{net}_i} = \frac{\partial E}{\partial \text{net}_i} = \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial \text{net}_i} = F_{-x_i} \ast s'(\text{net}_i) \]
\[ x_i = x_i, \quad 1 \leq i \leq m \]

\[ \text{net}_i = \sum_{j=1}^{i-1} W_{ij} x_i, \quad m < i \leq N + n \]

\[ x_i = s(\text{net}_i), \quad m < i \leq N + n \]

\[ Y_i = x_{i+N}, \quad 1 \leq i \leq n \]

\[ F \_ W_{ij} = \frac{\partial E}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} \frac{\partial \text{net}_i}{\partial W_{ij}} = \sum_{t=1}^{T} \frac{\partial E}{\partial \text{net}_i} * x_j \]

\[ = \sum_{t=1}^{T} F \_ \text{net}_i * x_j \]

\[ F \_ \text{net}_i = \frac{\partial E}{\partial \text{net}_i} = \frac{\partial E}{\partial x_i} \frac{\partial x_i}{\partial \text{net}_i} = F \_ x_i * s'(\text{net}_i) \]

\[ F \_ x_i = \frac{\partial E}{\partial \hat{Y}_{i-N}} + \sum_{j>i} \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial x_j} = F \_ \hat{Y}_{i-N} + \sum_{j>i} F \_ \text{net}_j W_{ji} \]
Pattern Learning

• Until satisfied, do
  – For each training data \( t = 1: T \)
    • Compute network outputs with feedforward algo
    • For each outputs \( i = 1: n \)
      – Compute \( F_\hat{Y}(t) = \frac{\partial E}{\partial \hat{Y}_i(t)} = \hat{Y}_i(t) - Y_i(t) \)
    • For each neurons in hidden layer
      – Compute \( F_{W_{ij}} \) with backpropagation
    • For all weights
      – Update \( W_{ij} = W_{ij} - \text{learning\_rate} * F_{W_{ij}} \)
Batch Learning

• Until satisfied, do
  – For each training data $t = 1: T$
    • Compute network outputs with feedforward algo
    • For each outputs $i = 1: n$
      – Compute $F_\hat{Y}(t) = \frac{\partial E}{\partial \hat{Y}_i(t)} = \hat{Y}_i(t) - Y_i(t)$
    • For each neurons in hidden layer
      – Compute $F_W_{ij}$ with backpropagation

  – For all weights
    • Update $W_{ij} = W_{ij} - \text{learning\_rate} \times F_W_{ij}$
Suggestions for implementation

• Until satisfied, do
  – For each training data \( t = 1: T \)
    • Compute network outputs with feedforward algo
    • For each outputs \( i = 1: n \)
      – Compute
    • For each neurons in hidden layer
      – Compute \( F_{W_{ij}} \) with backpropagation
  • For all weights
    – Update \( W_{ij} = W_{ij} - \text{learning_rate} \times F_{W_{ij}} \)

Change of error \( E \) or change of weight \( W \) converges

Range checking

Use pattern learning rather than batch learning
Backpropagation through time

• Now we want to account for what we saw earlier
• Example: speech recognition
• Every neuron at time t is allowed to input values from any of the neurons at time t-1 and t-2
Time = \( t \)

\[ \text{Time} = t \]

\[ \text{Time} = t+1 \]
Time = t

\[ W'_{ij} \]

Time = t+1
$F_{-x_i} = \frac{\partial E}{\partial \hat{Y}_{i-N}} + \sum_{j>i} \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial x_j}$

$= F_{\hat{Y}_{i-N}} + \sum_{j>i} F_{\text{net}_j} W_{ji}$
Time = $t$

\[
F_{x_i} = \frac{\partial E}{\partial \hat{Y}_{i-N}} + \sum_{j>i} \frac{\partial E}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial x_j}
\]

\[
= F_{\hat{Y}_{i-N}} + \sum_{j>i} F_{\text{net}_j} W_{ji}
\]

\[
F_{x_i}(t) = F_{\hat{Y}_{i-N}}(t) + \sum_{j>i} W_{ji} * F_{\text{net}_j}(t)
\]

\[
+ \sum_{j>i} W'_{ji} * F_{\text{net}_j}(t + 1)
\]
Notes on implementing Backpropagation through time

• We need to store a lot of intermediate information
• Use smaller learning rates
• Initialize $W'$ weights to zero
• We might not care about errors all the time
Other things to note

• Choice of network topology
  – If implementing a brain-like network (each neuron only receive input from a small number of other notes), use LinkedList

• Performance concerns
  – Avoid calculating exponential  \( s(z) = 1/(1 + e^{-z}) \)
  – Thresholding instead  
    \[
    s(z) = \begin{cases} 
    0, & z < 0 \\
    z, & 0 < z < 1 \\
    1, & z > 1.
    \end{cases}
    \]
Other Applications

• Extension to stochastic models
• Instead of having $Y(t)$ as output, we can have $U(t) = F(Y(t))$ as output
• Just apply one more chain rule when calculating derivatives