ECE 471/571 - Lecture 14
Back Propagation
03/22/17

Types of NN
- Recurrent (feedback during operation)
  - Hopfield
  - Kohonen
  - Associative memory
- Feedforward
  - No feedback during operation or testing (only during determination of weights or training)
  - Perceptron
  - Backpropagation

History
- In the 1980s, NN became fashionable again, after the dark age during the 1970s
- One of the reasons is the paper by Rumelhart, Hinton, and McClelland, which made the BP algorithm famous
- The algorithm was first discovered in 1960s, but didn’t draw much attention
- BP is most famous for applications for layered feedforward networks, or multilayer perceptrons
Limitations of Perceptron

- The output only has two values (1 or 0)
- Can only classify samples which are linearly separable (straight line or straight plane)
- Single layer: can only train AND, OR, NOT
- Can't train a network functions like XOR

XOR (3-layer NN)

\[ S_1, S_2 \text{ are identity functions} \]
\[ S_3, S_4, S_5 \text{ are sigmoid} \]
\[ w_{13} = 1.0, w_{14} = -1.0 \]
\[ w_{24} = 1.0, w_{23} = -1.0 \]
\[ w_{35} = 0.11, w_{45} = -0.1 \]

The input takes on only -1 and 1

BP – 3-Layer Network

\[ E = \frac{1}{2} \sum_j (T_j - S(y_j))^2 \]
Choose a set of initial \( \omega_{st} \)
\[ \omega_{st}^{t+1} = \omega_{st}^t - \epsilon^t \frac{\partial E^t}{\partial \omega_{st}} \]

The problem is essentially "how to choose weight \( \omega \) to minimize the error between the expected output and the actual output"
**Exercise**

\[
y_j = \sum_{q} S_q(h_q) = \frac{\partial y_j}{\partial S_q} = \omega_{qy} \text{ and } \frac{\partial y_j}{\partial \omega_{qy}} = S_q(h_q)
\]

\[
h_q = \sum_{x} x_\omega \omega_q = \frac{\partial h_q}{\partial x_i} = \omega_{iq} \text{ and } \frac{\partial h_q}{\partial \omega_{iq}} = x_i
\]

---

**The Derivative – Chain Rule**

\[
\Delta \omega_q = -\frac{\partial E}{\partial \omega_q} = -\frac{\partial E}{\partial S_j} \frac{\partial h_q}{\partial \omega_q} = -(T - S_j)(S_j(h_q))
\]

\[
\Delta \omega_q = -\frac{\partial E}{\partial \omega_q} = \sum \frac{\partial E}{\partial S_j} \frac{\partial h_q}{\partial \omega_q} \frac{\partial \omega_q}{\partial \omega_q}
\]

\[
= \sum (T - S_j)[S_j(h_q)]d\omega_q
\]

---

**Threshold Function**

- Traditional threshold function as proposed by McCulloch-Pitts is binary function
- The importance of differentiable
- A threshold-like but differentiable form for \( S \) (25 years)
- The sigmoid

\[
S(x) = \frac{1}{1 + \exp(-x)}
\]
Practical Improvements to Backpropagation

Activation (Threshold) Function

- The signum function
  \[ S(x) = \text{signum}(x) = \begin{cases} 1 & \text{if } x \neq 0 \\ -1 & \text{if } x < 0 \end{cases} \]

- The sigmoid function
  \[ S(x) = \text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \]
  - Nonlinear
  - Saturate
  - Continuity and smoothness
  - Monotonicity (so \( S'(x) > 0 \))
  - Improved
    - Centered at zero
    - Antisymmetric (odd) – leads to faster learning
    - A = 1.716, b = 2/3, to keep \( S'(0) \to 1 \), the linear range is \(-1 < x < 1\), and the extrema of \( S''(x) \) occur roughly at \( x \to \pm 2 \)
Data Standardization

- Problem in the units of the inputs
  - Different units cause magnitude of difference
  - Same units cause magnitude of difference
- Standardization – scaling input
  - Shift the input pattern
    - The average over the training set of each feature is zero
  - Scale the full data set
    - Have the same variance in each feature component (around 1.0)

Target Values (output)

- Instead of one-of-c (c is the number of classes), we use +1/-1
  - +1 indicates target category
  - -1 indicates non-target category
- For faster convergence
Number of Hidden Layers

- The number of hidden layers governs the expressive power of the network, and also the complexity of the decision boundary.
- More hidden layers -> higher expressive power -> better tuned to the particular training set -> poor performance on the testing set.
- Rule of thumb:
  - Choose the number of weights to be roughly \( n/10 \), where \( n \) is the total number of samples in the training set.
  - Start with a “large” number of hidden units, and “decay”, prune, or eliminate weights.

Rule of thumb

- Choose the number of weights to be roughly \( n/10 \), where \( n \) is the total number of samples in the training set.
- Start with a “large” number of hidden units, and “decay”, prune, or eliminate weights.

Number of Hidden Layers

- Fast and uniform learning
  - All weights reach their final equilibrium values at about the same time.
  - Choose weights randomly from a uniform distribution to help ensure uniform learning.
  - Equal negative and positive weights.
  - Set the weights such that the integration value at a hidden unit is in the range of \(-1\) and \(+1\).
  - Input-to-hidden weights: \((-1/\sqrt{d}), 1/\sqrt{d})\).
  - Hidden-to-output weights: \((-1/\sqrt{n_h}), 1/\sqrt{n_h})\), \(n_h\) is the number of connected units.

Initializing Weight

- Can’t start with zero.
- Fast and uniform learning.
  - All weights reach their final equilibrium values at about the same time.
  - Choose weights randomly from a uniform distribution to help ensure uniform learning.
  - Equal negative and positive weights.
  - Set the weights such that the integration value at a hidden unit is in the range of \(-1\) and \(+1\).
  - Input-to-hidden weights: \((-1/\sqrt{d}), 1/\sqrt{d})\).
  - Hidden-to-output weights: \((-1/\sqrt{n_h}), 1/\sqrt{n_h})\), \(n_h\) is the number of connected units.
**Learning Rate**

- The optimal learning rate
  - Calculate the 2nd derivative of the objective function with respect to each weight
  - Set the optimal learning rate separately for each weight
  - A learning rate of 0.1 is often adequate
  - The maximum learning rate is $c_{\text{max}} < 2c_{\text{opt}}$
  - When $c_{\text{opt}} < c < 2c_{\text{opt}}$, the convergence is slow

**Plateaus or Flat Surface in $S'$**

- Plateaus
  - Regions where the derivative $\frac{\partial E}{\partial \omega}$ is very small
  - When the sigmoid function saturates
- Momentum
  - Allows the network to learn more quickly when plateaus in the error surface exist
  \[
  \omega_{st}^{k+1} = \omega_{st}^k - c_k \frac{\partial E^k}{\partial \omega}
  \]
  \[
  \omega_{st}^{k+1} = \omega_{st}^k + (1 - c^k) \Delta \omega_k + c^k (\omega_{st}^k - \omega_{st}^{k-1})
  \]
**Weight Decay**

- Should almost always lead to improved performance

\[ \omega^{new} = \omega^{old} (1 - \epsilon) \]

**Batch Training vs. On-line Training**

- **Batch training**
  - Add up the weight changes for all the training patterns and apply them in one go
  - GD

- **On-line training**
  - Update all the weights immediately after processing each training pattern
  - Not true GD but faster learning rate

**Other Improvements**

- Other error function (Minkowski error)