ROB 537: Learning-Based Control

Week 1, Lecture 2

Multi-Layered Feed Forward Neural Networks

Announcements:
HW 1 Due on 10/2

Single Layer Feed Forward Networks

- Error on pattern n: \( e^n = t^n - y^n \)

- Mean Square Error: \( E = \frac{1}{2} \sum_{n=1}^{N} (e^n)^2 \)

- Gradient descent: \( \Delta w = -\eta \frac{\partial E}{\partial w} \)

We’ll drop the \( n \) superscript from \( x, y, t, e \) for simplicity
Single Layer Feed Forward Networks

- Differentiate error with respect to weights

\[
\frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial z_i} \frac{\partial z_i}{\partial w_{i,j}}
\]

where \( z_i = \sum_j w_{i,j} x_j \)

Single Layer Feed Forward Networks

- Compute each of the partials:

  - First term:
    \[
    \frac{\partial E}{\partial e_i} = \frac{\partial}{\partial e_i} \left( \frac{1}{2} e_i^2 \right) = e_i
    \]

  - Second term:
    \[
    \frac{\partial e_i}{\partial y_i} = \frac{\partial (t_i - y_i)}{\partial y_i} = -1
    \]

  - Fourth term:
    \[
    \frac{\partial x_j}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}} \left( \sum_j w_{i,j} x_j \right) = x_j
    \]
Single Layer Feed Forward Networks

• Third term:

\[
\frac{\partial y_i}{\partial z_i} = \frac{\partial}{\partial z_i} \left( \frac{1}{1 + e^{-z_i}} \right)
\]

\[
= \left( \frac{1}{1 + e^{-z_i}} \right)^2 e^{-z_i} \left( -1 \right)
\]

\[
= \frac{1}{1 + e^{-z_i}} \left( \frac{e^{-z_i}}{1 + e^{-z_i}} \right)
\]

\[
= \frac{1}{1 + e^{-z_i}} \left( \frac{1 + e^{-z_i} - 1}{1 + e^{-z_i}} \right)
\]

\[
= y_i \left( 1 - y_i \right)
\]
Single Layer Feed Forward Networks

- Sigmoid activation function:
  \[ f(a) = \frac{1}{1 + e^{-a}} \]

- Gradient descent:
  \[
  \frac{\partial E}{\partial w_{i,j}} = \frac{\partial E}{\partial e_i} \frac{\partial e_i}{\partial y_i} \frac{\partial y_i}{\partial z_i} \frac{\partial z_i}{\partial w_{i,j}}
  \]
  \[= e_i (-1) y_i (1 - y_i) (x_j) \]

- Weight update:
  \[\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}} = \eta e_i y_i (1 - y_i) x_j\]

Illustration of Gradient Descent

![Illustration of Gradient Descent](image)
Multi Layer Feed Forward Networks

\[ y_k = f \left( \sum_j w_{j,k} h_j + w_{0,k} \right) \]

\[ f(a) = \frac{1}{1 + e^{-a}} \]

\[ = f \left( \sum_j w_{j,k} \cdot f \left( \sum_i v_{i,j} x_i + v_{0,j} \right) + w_{0,k} \right) \]
Weight updates

- Derivative of error wrt weight j,k:
  \[
  \frac{\partial E}{\partial w_{j,k}} = e_k^e \frac{\partial e_k^e}{\partial w_{j,k}} \\
  = -e_k^e \frac{\partial f_k^o}{\partial w_{j,k}} \\
  = -e_k^e f \left( \sum_j w_{j,k} h_j^o \right) \left( 1 - f \left( \sum_j w_{j,k} h_j^o \right) \right) h_j \\
  = -e_k^e y_k^o (1 - y_k^o) h_j
  \]
Multi Layer Feed Forward Networks

What are the errors for the hidden layer?

We don’t know the targets.

Now what?

\[ h_j = f\left(\sum_i v_{i,j}x_i + v_{0,j}\right) \]

Error Backpropagation

- Updating input-hidden layer weights:
  \[ \Delta v_{i,j} = \eta \delta_j x_i \]

- Delta:
  \[ \delta_j = e_j^n f\left(\sum_i v_{i,j}x_i^n\right)\left(1 - f\left(\sum_i v_{i,j}x_i^n\right)\right) \]
  \[ = e_j^n h_j^n(1 - h_j^n) \]

Errors for the hidden layer:

Backpropagated deltas from output layer
Backpropagation Summary:

- For sigmoidal activation functions, update any weight connecting “input” \( i \) to “output” \( j \):

\[
\Delta w_{i,j} = \eta \delta_j x_i
\]

- Deltas given by:
  - For output layer
    \[
    \delta_j = e_j y_j (1 - y_j)
    \]
  - For hidden layer
    \[
    \delta_j = \sum_k w_{j,k} \delta_k \cdot h_j (1 - h_j)
    \]

Backpropagation Algorithm

- For each epoch:
  - Present pattern \( x \) to network
    - Propagate signal forward:
      - Compute hidden unit values
      - Compute output value
    - Find error
      - Compute output layer deltas
      - Compute hidden layer deltas
    - Compute gradient for each weight
    - Update each weight
  - Present next pattern
- Repeat this process until MSE is satisfactory
Feed Forward Neural Net Recap

- A multi-layered FFNN with sigmoidal activation functions is also called a multi-layered Perceptron.

- Key characteristic:
  - Global processing
  - All hidden units react to all inputs

- What if we want local processing?
  - Some hidden units react to some inputs?

Radial Basis Function Networks

Key RBF differences:
- Local activation
- Linear output layer recommended
- Euclidean norm activation
- All hidden units are different functions
- One “hidden” layer
Radial Basis Function Networks

- $c_j$ is the “center” of the $j$th radial basis function
  
  \[ c_j = \{c_{j,1}, \ldots, c_{j,p}, \ldots, c_{j,N}\} \]

- $\sigma_j$ is the “radius” of the $j$th radial basis function

\[ R_j(x) = \exp\left(-\frac{|x - c_j|^2}{2(\sigma_j)^2}\right) \]
Radial Basis Function Networks

\[ R_j(x) = \exp \left( -\frac{|x - c_j|^2}{2\sigma_j^2} \right) \]

\[ y_k = f \left( \sum_j w_{j,k} R_j + w_{0,k} \right) \]

\[ = f \left( \sum_j w_{j,k} \cdot \exp \left( -\frac{|x - c_j|^2}{2\sigma_j^2} \right) + w_{0,k} \right) \]

RBF Center Updates

- For single output (y) linear output layer:

\[ \frac{\partial E}{\partial c_{j,k}} = \frac{\partial E}{\partial e} \frac{\partial e}{\partial y} \frac{\partial y}{\partial R_j} \frac{\partial R_j}{\partial c_{j,k}} \]

- Write set of partials:

- What is \( \frac{\partial R_j}{\partial c_{j,k}} \) ?
RBF Center Updates

- For quadratic distance:

\[ |x - c_j|^2 = (x - c_{j,1})^2 + \cdots + (x - c_{j,k})^2 + \cdots + (x - c_{j,N})^2 \]

- Center updates:

\[
\frac{\partial R_j}{\partial c_{j,k}} = \frac{\partial \exp \left( -\frac{|x - c_j|^2}{2(\sigma_j)^2} \right)}{\partial c_{j,k}}
\]

\[ = \exp \left( -\frac{|x - c_j|^2}{2(\sigma_j)^2} \right) (-2)(-1) \left( \frac{1}{2(\sigma_j)^2} \right) (x_k - c_{j,k}) \]

\[ = R_j \frac{(x_k - c_{j,k})}{(\sigma_j)^2} \]

RBF Center Updates

- For single output (y) linear output layer:

\[
\frac{\partial E}{\partial c_{j,k}} = \frac{\partial E}{\partial e} \frac{\partial e}{\partial y} \frac{\partial y}{\partial R_j} \frac{\partial R_j}{\partial c_{j,k}}
\]

\[ = e(-1) w_j \frac{\partial R_j}{\partial c_{j,k}} \]

\[ = e(-1) w_j R_j \frac{(x_k - c_{j,k})}{(\sigma_j)^2} \]

- Updating Centers:

\[
\Delta c_{j,k} = -\eta \frac{\partial E}{\partial c_{j,k}} = \eta e w_j R_j \frac{(x_k - c_{j,k})}{(\sigma_j)^2}
\]
Implementation issues

- Training, Testing and Validation
- Network architecture and Training
- Initial weights & Parameter selection
- Local minima
- Momentum term for weights
- Network complexity
- Convergence
- Generalization
- Model complexity
- Universal approximator theorem

Training, Testing and Validation

- Training: using known samples to set parameters
- Testing: Verifying that learned mapping applies to unseen samples
- Validation: “testing” on the training samples to set parameters
- Generalization: Ability to extend learning to new samples
- Example: 1000 data points
  - Use 600 for training: set parameters
  - Use 200 for validation: check performance, adjust parameters
  - Use 200 for testing: Generalization performance
- Cross-validation: Train and validate on data partitions.
  - 4-fold cross validation means split data into four and train on three quarters and validate on one fourth for each combination
  - All training data used for training (200 data points not used above)
Network Architecture and Training

- Architecture:
  - Feed forward network
  - 2 Layer FFN

- Neuron selection
  - Activation functions

- Learning Algorithm
  - Gradient descent

- How many hidden units?

- How long to train?

Initial Weights

- Initial Weights
  - Random
  - Seed special concept

- Clustering (for RBF networks)
Local Minima

- Can have multiple local minima
- Gradient descent goes to the closest local minimum:
  - solution: random restarts from multiple places in weight space

Momentum Term

- Weight update changes too fast with:
  \[ \Delta w_{i,j} = \eta \delta_j x_i \]
- Let each update be closer to last update. Give the gradient "momentum":
  \[ \Delta w''_{i,j} = \alpha \Delta w'_{i,j} + \eta \delta_j^n x_i^n \]
Convergence

- Preset time:
  - Train for 2000 epochs

- Preset error criteria:
  - Train until MSE reaches

- Relative error criteria:
  - Train till MSE changes by less than .1% per epoch

- Use some “left out” patterns to validate training. When validation error bottoms out, stop training.

Generalization

- Training set error reduced continuously
- Test set error (generalization error) increases after a point
  - Network starts to learn the “noise” in the training data
Model Complexity

Universal Function Approximation

• How good an approximator is a multi layer feed forward neural network?

Universal Approximation Theorem:

Under mild assumptions, for any given constant $\varepsilon$ and continuous function $f(x_1, \ldots, x_m)$, there exists a one hidden-layer MLP (F):

$$F(x_1, \ldots, x_m) = \Sigma_i v_i h(\Sigma_j w_{ij} x_j + b_j)$$

$h(\cdot)$ is nonlinear activation function

with the property that:

$$|f(x_1, \ldots, x_m) - F(x_1, \ldots, x_m)| < \varepsilon$$

Cybenko 1989; Hornik 1991
Video: Character Recognition

http://www.youtube.com/watch?v=3WR5pAa2m2E

Video: Speed Sign Recognition

http://www.youtube.com/watch?v=kkha3sPoU70