Today

- Continue Multilayer Neural Networks (MNN)
  - Review MNN structure
  - Backpropagation
  - Training Protocols
MNN: Feed Forward Operation

**input layer:** $d$ features

**hidden layer:**

**output layer:** $m$ outputs, one for each class

bias unit
MNN: Notation for Weights

- Use $w_{ji}$ to denote the weight between input unit $i$ and hidden unit $j$

  - Input unit $i$
  - Hidden unit $j$
  - $x^{(i)}$ to hidden unit $j$ via $w_{ji}$
  - $w_{ji}x^{(i)}$
  - Hidden unit $j$ to output unit $y_j$

- Use $v_{kj}$ to denote the weight between hidden unit $j$ and output unit $k$

  - Hidden unit $j$ to output unit $k$
  - $y_j$ to output unit $k$ via $v_{kj}$
  - $v_{kj}y_j$
MNN: Notation for Activation

- Use $net_j$ to denote the activation and hidden unit $j$
  
  $$net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0}$$

- Use $net^*_k$ to denote the activation at output unit $k$
  
  $$net^*_k = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0}$$
**Discriminant Function**

- Discriminant function for class $k$ (the output of the $k$th output unit)

$$g_k(x) = z_k = f \left( \sum_{j=1}^{N_H} v_{kj} f \left( \sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0} \right) + v_{k0} \right)$$

- Rich expressive power: every continuous discriminant function can be implemented with enough hidden units, 1 hidden layer, and proper nonlinear activation functions.
FIGURE 6.2. A 2-4-1 network (with bias) along with the response functions at different units; each hidden output unit has sigmoidal activation function $f(\cdot)$. In the case shown, the hidden unit outputs are paired in opposition thereby producing a “bump” at the output unit. Given a sufficiently large number of hidden units, any continuous function from input to output can be approximated arbitrarily well by such a network. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.
MNN Activation function

- Must be nonlinear for expressive power larger than that of perceptron
  - If use linear activation function at hidden layer, can only deal with linearly separable classes
  - Suppose at hidden unit $j$, $h(u) = a_i u$

$$g_k(x) = f \left( \sum_{j=1}^{N_H} v_{kj} h \left( \sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0} \right) + v_{k0} \right)$$

$$= f \left( \sum_{j=1}^{N_H} v_{kj} a_j \left( \sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0} \right) + v_{k0} \right)$$

$$= f \left( \sum_{i=1}^{d} \sum_{j=1}^{N_H} \left( v_{kj} a_j w_{ji} x^{(i)} + w_{j0} \right) + v_{k0} \right)$$

$$= f \left( \sum_{i=1}^{d} \sum_{j=1}^{N_H} \begin{pmatrix} w_i^{new} \\ v_{kj} a_j w_{ji} \end{pmatrix} + \begin{pmatrix} w_0^{new} \\ \sum_{j=1}^{N_H} w_{j0} + v_{k0} \end{pmatrix} \right)$$
MNN Activation function

- In previous example, used discontinuous activation function

\[ f(\text{net}_k) = \begin{cases} 
1 & \text{if } \text{net}_k \geq 0 \\
-1 & \text{if } \text{net}_k < 0
\end{cases} \]

- We will use gradient descent for learning, so we need to use continuous activation function

- From now on, assume \( f \) is a differentiable function
**MNN: Modes of Operation**

- Network have two modes of operation:

  - **Feedforward**
    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)

  - **Learning**
    The supervised learning consists of presenting an input pattern and modifying the network parameters (weights) to reduce distances between the computed output and the desired output
**MNN: Class Representation**

- Training samples \( x_1, \ldots, x_n \) each of class \( 1, \ldots, m \)
- Let network output \( z \) represent class \( c \) as *target* \( t^{(c)} \)

\[
\begin{bmatrix}
z_1 \\
\vdots \\
z_c \\
\vdots \\
z_m \\
\end{bmatrix} = t^{(c)} = 
\begin{bmatrix}
0 \\
\vdots \\
1 \\
\vdots \\
0 \\
\end{bmatrix}
\]

**Our Ultimate Goal For FeedForward Operation**

Sample of class \( c \) \( \xrightarrow{\text{MNN with weights } w_{ji} \text{ and } v_{kj}} \) \( t^{(c)} \)

**MNN training to achieve the Ultimate Goal**

Modify (learn) MNN parameters \( w_{ji} \) and \( v_{kj} \) so that for each *training* sample of class \( c \) MNN output \( z = t^{(c)} \)
1. Initialize weights $w_{ji}$ and $v_{kj}$ randomly
2. Iterate until a stopping criterion is reached

Choose $p$

Input sample $x_p$

MNN with weights $w_{ji}$ and $v_{kj}$

Output $z = \begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}$

Compare output $z$ with the desired target $t$; adjust $w_{ji}$ and $v_{kj}$ to move closer to the goal $t$ (by backpropagation)
BackPropagation

- Learn $w_{ji}$ and $v_{kj}$ by minimizing the training error
- What is the training error?
- Suppose the output of MNN for sample $x$ is $z$ and the target (desired output for $x$) is $t$
- Error on one sample: $J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2$
- Training error: $J(w, v) = \frac{1}{2n} \sum_{i=1}^{n} \sum_{c=1}^{m} (t_c^{(i)} - z_c^{(i)})^2$
- Use gradient descent:
  
  $v^{(0)}, w^{(0)} = \text{random}$
  repeat until convergence:
  
  $w^{(t+1)} = w^{(t)} - \eta \nabla_w J(w^{(t)})$
  $v^{(t+1)} = v^{(t)} - \eta \nabla_v J(v^{(t)})$
**BackPropagation**

- For simplicity, first take training error for one sample $x_i$

$$J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2$$

Need to compute

1. partial derivative w.r.t. hidden-to-output weights $\frac{\partial J}{\partial v_{kj}}$

2. partial derivative w.r.t. input-to-hidden weights $\frac{\partial J}{\partial w_{ji}}$

$$z_k = f \left( \sum_{j=1}^{NH} v_{kj} f \left( \sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0} \right) + v_{k0} \right)$$
BackPropagation: Layered Model

activation at hidden unit $j$

\[ net_j = \sum_{i=1}^{d} x^{(i)} w_{ji} + w_{j0} \]

output at hidden unit $j$

\[ y_j = f(net_j) \]

activation at output unit $k$

\[ net_k^* = \sum_{j=1}^{N_H} y_j v_{kj} + v_{k0} \]

activation at output unit $k$

\[ z_k = f(net_k^*) \]

objective function

\[ J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2 \]

chain rule

\[ \frac{\partial J}{\partial v_{kj}} \]

chain rule

\[ \frac{\partial J}{\partial w_{ji}} \]
BackPropagation

\[ net_k = \sum_{j=1}^{N_h} y_j v_{kj} + v_{k0} \quad \Rightarrow \quad z_k = f(net_k^*) \quad \Rightarrow \quad J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2 \]

- First compute hidden-to-output derivatives

\[
\frac{\partial J}{\partial v_{kj}} = \frac{1}{2} \sum_{c=1}^{m} \frac{\partial}{\partial v_{kj}} (t_c - z_c)^2 = \sum_{c=1}^{m} (t_c - z_c) \frac{\partial}{\partial v_{kj}} (t_c - z_c)
\]

\[
= (t_k - z_k) \frac{\partial}{\partial v_{kj}} (t_k - z_k) = -(t_k - z_k) \frac{\partial}{\partial v_{kj}} (z_k)
\]

\[
= -(t_k - z_k) \frac{\partial z_k}{\partial net_k^*} \frac{\partial net_k^*}{\partial v_{kj}} = 
\]

\[
= \begin{cases} 
-(t_k - z_k) f'(net_k^*) y_j & \text{if } j \neq 0 \\
-(t_k - z_k) f'(net_k^*) & \text{if } j = 0
\end{cases}
\]
Gradient Descent *Single Sample* Update Rule for hidden-to-output weights $v_{kj}$

\[
\begin{align*}
\text{j > 0: } & \quad v_{kj}^{(t+1)} = v_{kj}^{(t)} + \eta (t_k - z_k) f'(net_k^*) y_j \\
\text{j = 0 (bias weight): } & \quad v_{k0}^{(t+1)} = v_{k0}^{(t)} + \eta (t_k - z_k) f'(net_k^*)
\end{align*}
\]
BackPropagation

Now compute input-to-hidden \( \frac{\partial J}{\partial w_{ji}} \)

\[
\frac{\partial J}{\partial w_{ji}} = \sum_{k=1}^{m} (t_k - z_k) \frac{\partial}{\partial w_{ji}} (t_k - z_k)
\]

\[
= -\sum_{k=1}^{m} (t_k - z_k) \frac{\partial z_k}{\partial w_{ji}} = -\sum_{k=1}^{m} (t_k - z_k) \frac{\partial z_k}{\partial net_k^*} \frac{\partial net_k^*}{\partial w_{ji}}
\]

\[
= -\sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) \frac{\partial net_k^*}{\partial y_j} \frac{\partial y_j}{\partial w_{ji}}
\]

\[
= -\sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}
\]

\[
= -\sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}
\]

\[
= \begin{cases} 
  -\sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj} f'(net_j) x^{(i)} & \text{if } i \neq 0 \\
  -\sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj} f'(net_j) & \text{if } i = 0
\end{cases}
\]

\[
net_h = \sum_{h=1}^{d} x^{(i)} w_{hi} + w_{h0}
\]

\[
y_j = f(net_j)
\]

\[
net_k^* = \sum_{s=1}^{N_h} y_s v_{ks} + v_{k0}
\]

\[
z_k = f(net_k^*)
\]

\[
J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2
\]
BackPropagation

\[
\frac{\partial J}{\partial w_{ji}} = \begin{cases} 
-f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net^*_k) v_{kj} & \text{if } i \neq 0 \\
-f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net^*_k) v_{kj} & \text{if } i = 0
\end{cases}
\]

Gradient Descent Single Sample Update Rule for input-to-hidden weights \( w_{ji} \)

\[
i > 0: \quad w_{ji}^{(t+1)} = w_{ji}^{(t)} + \eta f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net^*_k) v_{kj}
\]

\[
i = 0 \text{ (bias weight)}: \quad w_{j0}^{(t+1)} = w_{j0}^{(t)} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net^*_k) v_{kj}
\]
BackPropagation of Errors

\[
\frac{\partial J}{\partial w_{ji}} = -f'(net_j)x^{(i)}\sum_{k=1}^{m}(t_k - z_k)f'(net^*_k)v_{kj} \\
\frac{\partial J}{\partial v_{kj}} = -(t_k - z_k)f'(net^*_k)y_j
\]

- Name “backpropagation” because during training, errors propagated back from output to hidden layer
Consider update rule for hidden-to-output weights:

\[ v_{kj}^{(t+1)} = v_{kj}^{(t)} + \eta(t_k - z_k)f'(net_k^*)y_j \]

Suppose \( t_k - z_k > 0 \)

Then output of the \( k \)th hidden unit is too small: \( t_k > z_k \)

Typically activation function \( f \) is s.t. \( f' > 0 \)

Thus \( (t_k - z_k)f'(net_k^*) > 0 \)

There are 2 cases:

1. \( y_j > 0 \), then to increase \( z_k \), should increase weight \( v_{kj} \)
   which is exactly what we do since \( \eta(t_k - z_k)f'(net_k^*)y_j > 0 \)

2. \( y_j < 0 \), then to increase \( z_k \), should decrease weight \( v_{kj} \)
   which is exactly what we do since \( \eta(t_k - z_k)f'(net_k^*)y_j < 0 \)
BackPropagation

- The case \( t_k - z_k < 0 \) is analogous

- Similarly, can show that input-to-hidden weights make sense

- Important: weights should be initialized to random \textit{nonzero} numbers

\[
\frac{\partial J}{\partial w_{ji}} = -f'(net_j)x^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}
\]

- if \( v_{kj} = 0 \), input-to-hidden weights \( w_{ji} \) never updated
Training Protocols

- How to present samples in training set and update the weights?

- Three major training protocols:
  1. Stochastic
     - Patterns are chosen randomly from the training set, and network weights are updated after every sample presentation
  2. Batch
     - weights are update based on all samples; iterate weight update
  3. Online
     - each sample is presented only once, weight update after each sample presentation
Stochastic Back Propagation

1. Initialize
   - number of hidden layers $n_H$
   - weights $w, v$
   - convergence criterion $\theta$ and learning rate $\eta$
   - time $t = 0$

2. do
   - $x \leftarrow$ randomly chosen training pattern
   for all  $0 \leq i \leq d$,  $0 \leq j \leq n_H$,  $0 \leq k \leq m$
     
     $w_{ji} = w_{ji} + \eta f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$
     
     $w_{j0} = w_{j0} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$
     
     $v_{kj} = v_{kj} + \eta(t_k - z_k) f'(net_k^*) y_j$
     
     $v_{k0} = v_{k0} + \eta(t_k - z_k) f'(net_k^*)$
     
   $t = t + 1$
   until  $\|J\| < \theta$

3. return  $v, w$
**Batch Back Propagation**

- This is the *true* gradient descent, (unlike stochastic propagation)
- For simplicity, derived backpropagation for a single sample objective function:
  \[ J(w, v) = \frac{1}{2} \sum_{c=1}^{m} (t_c - z_c)^2 \]
- The full objective function:
  \[ J(w, v) = \frac{1}{2} \sum_{i=1}^{n} \sum_{c=1}^{m} (t_c^{(i)} - z_c^{(i)})^2 \]
- Derivative of full objective function is just a sum of derivatives for each sample:
  \[ \frac{\partial}{\partial w} J(w, v) = \frac{1}{2} \sum_{i=1}^{n} \frac{\partial}{\partial w} \left( \sum_{c=1}^{m} (t_c^{(i)} - z_c^{(i)})^2 \right) \]

already derived this
Batch Back Propagation

- For example,

\[
\frac{\partial J}{\partial w_{ji}} = \sum_{p=1}^{n} - f'(net_j) x^{(i)}_{p} \sum_{k=1}^{m} (t_k - z_k) f'(net^*_k) v_{kj}
\]
Batch Back Propagation

1. Initialize $n_H$, $w$, $v$, $\theta$, $\eta$, $t = 0$

2. \(\text{do}\)

   \[ \begin{align*}
   \Delta v_{kj} &= \Delta v_{k0} = \Delta w_{ji} = \Delta w_{j0} = 0 \\
   \text{for all } 1 \leq p \leq n \\
   \text{for all } 0 \leq i \leq d, \ 0 \leq j \leq n_H, \ 0 \leq k \leq m \\
   \Delta v_{kj} &= \Delta v_{kj} + \eta(t_k - z_k)f'(net_k^*)y_j \\
   \Delta v_{k0} &= \Delta v_{k0} + \eta(t_k - z_k)f'(net_k^*) \\
   \Delta w_{ji} &= \Delta w_{ji} + \eta f'(net_j^*)x_p^{(i)} \sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj} \\
   \Delta w_{j0} &= \Delta w_{j0} + \eta f'(net_j^*) \sum_{k=1}^{m}(t_k - z_k)f'(net_k^*)v_{kj} \\
   \end{align*} \]

   \[ v_{kj} = v_{kj} + \Delta v_{kj}; \quad v_{k0} = v_{k0} + \Delta v_{k0}; \quad w_{ji} = w_{ji} + \Delta w_{ji}; \quad w_{j0} = w_{j0} + \Delta w_{j0} \]

   \(t = t + 1\)

3. \(\text{return } v, w\)
Training Protocols

1. Batch
   - True gradient descent
2. Stochastic
   - Faster than batch method
   - Usually the recommended way
3. Online
   - Used when number of samples is so large it does not fit in the memory
   - Dependent on the order of sample presentation
   - Should be avoided when possible