CS434b/654b: Pattern Recognition
Prof. Olga Veksler

Lecture 12

Multilayer Neural Networks

Brain vs. Computer

- Designed to solve logic and arithmetic problems
- Can solve a gazillion arithmetic and logic problems in an hour
- absolute precision
- Usually one very fast procesor
- high reliability

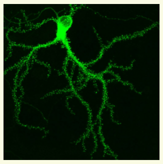
- Evolved (in a large part) for pattern recognition
- Can solve a gazillion of PR problems in an hour
- Huge number of parallel but relatively slow and unreliable processors
- not perfectly precise
- not perfectly reliable

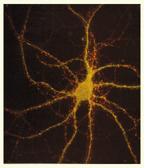
Seek an inspiration from human brain for PR?

Today

- Multilayer Neural Networks
  - Inspiration from Biology
  - History
  - Perceptron
  - Multilayer perceptron

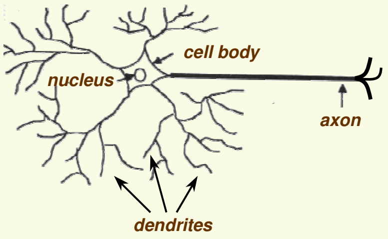
Neuron: Basic Brain Processor





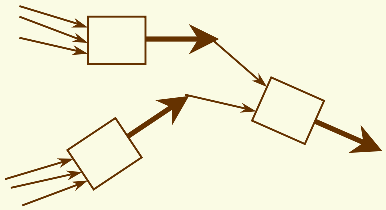
- Neurons are nerve cells that transmit signals to and from brains at the speed of around 200mph
- Each neuron cell communicates to anywhere from 1000 to 10,000 other neurons, muscle cells, glands, so on
- Have around ***10^{10}*** neurons in our brain (network of neurons)
- Most neurons a person is ever going to have are already present at birth

Neuron: Basic Brain Processor

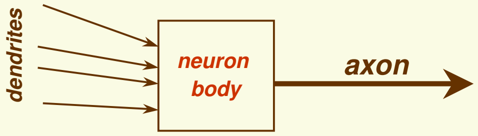


- Main components of a neuron
  - ***Cell body*** which holds DNA information in ***nucleus***
  - ***Dendrites*** may have thousands of dendrites, usually short
  - ***axon*** long structure, which splits in possibly thousands branches at the end. May be up to 1 meter long

Neural Network



Neuron in Action (simplified)



- ***Input*** : neuron collects signals from other neurons through dendrites, may have thousands of dendrites
- ***Processor:*** Signals are accumulated and processed by the cell body
- ***Output:*** If the strength of incoming signals is large enough, the cell body sends a signal (a spike of electrical activity) to the axon

ANN History: Birth

- 1943, famous paper by W. McCulloch (neurophysiologist) and W. Pitts (mathematician)
  - Using only math and algorithms, constructed a model of how neural network may work
  - Showed it is possible to construct any computable function with their network
  - Was it possible to make a model of thoughts of a human being?
  - Considered to be the birth of AI
- 1949, D. Hebb, introduced the first (purely pshychological) theory of learning
  - Brain learns at tasks through life, thereby it goes through tremendous changes
  - If two neurons fire together, they strengthen each other's responses and are likely to fire together in the future

ANN History: First Successes

- 1958, F. Rosenblatt,
  - perceptron, oldest neural network still in use today
  - Algorithm to train the perceptron network (training is still the most actively researched area today)
  - Built in hardware
  - Proved convergence in linearly separable case
- 1959, B. Widrow and M. Hoff
  - Madaline
  - First ANN applied to real problem (eliminate echoes in phone lines)
  - Still in commercial use

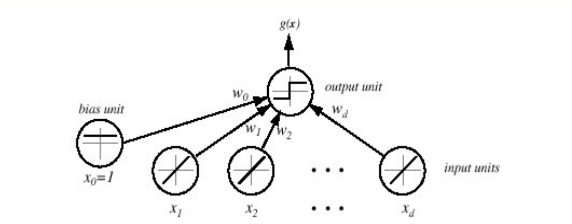
ANN History: Revival

- Revival of ANN in 1980's
- 1982, J. Hopfield
  - New kind of networks (Hopfield's networks)
  - Bidirectional connections between neurons
  - Implements associative memory
- 1982 joint US-Japanese conference on ANN
  - US worries that it will stay behind
- Many examples of mulitlayer NN appear
- 1982, discovery of backpropagation algorithm
  - Allows a network to learn not linearly separable classes
  - Discovered independently by
    1. Y. Lecunn
    2. D. Parker
    3. Rumelhart, Hinton, Williams

ANN History: Stagnation

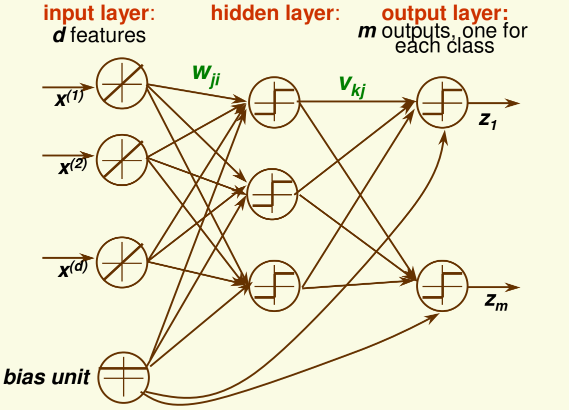
- Early success lead to a lot of claims which were not fulfilled
- 1969, M. Minsky and S. Pappert
  - Book "Perceptrons"
  - Proved that perceptrons can learn only linearly separable classes
  - In particular cannot learn very simple XOR function
  - Conjectured that multilayer neural networks also limited by linearly separable functions
- No funding and almost no research (at least in North America) in 1970's as the result of 2 things above

ANN: Perceptron



- Input and output layers
- $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + \boldsymbol{w}_0$
- Limitation: can learn only linearly separable classes

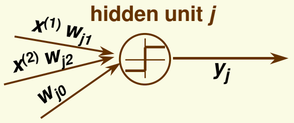
MNN: Feed Forward Operation



MNN: Notation for Activation

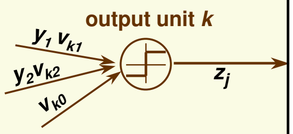
- Use ***net***$_j$ to denote the activation and hidden unit ***j***

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



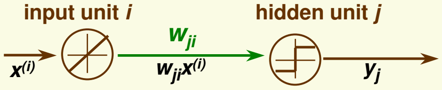
- Use ***net****$_k$ to denote the activation at output unit ***k***

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

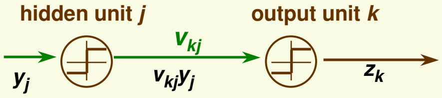


MNN: Notation for Weights

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



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



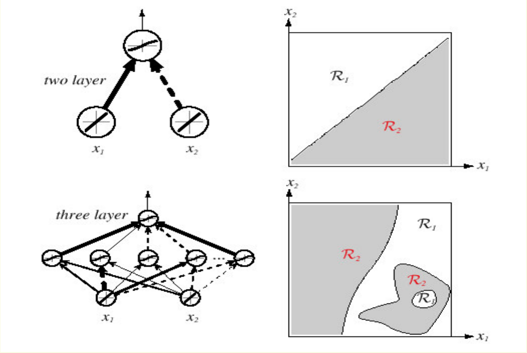
Discriminant Function

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

$$g_k(\mathbf{x}) = z_k =$$

$$= \underbrace{f\left(\sum_{j=1}^{N_H} v_{kj} f\left(\overbrace{\sum_{i=1}^{d} w_{ji} x^{(i)} + w_{j0}}^{\text{activation at jth hidden unit}}\right) + v_{k0}\right)}_{\text{activation at kth output unit}}$$

Discriminant Function



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***$_j$***u***

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

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

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

$$= f\left(\sum_{i=1}^{d} x^{(i)} \overset{w_i^{new}}{\sum_{j=1}^{N_H} v_{kj} a_j w_{ji}} + \overset{w_0^{new}}{\left(\sum_{j=1}^{N_H} v_{kj} a_j w_{j0} + v_{k0}\right)}\right)$$

Expressive Power of MNN

- It can be shown that every **continuous** function from input to output can be implemented with enough hidden units, 1 hidden layer, and proper nonlinear activation functions
- This is more of theoretical than practical interest
  - The proof is not constructive (does not tell us exactly how to construct the MNN)
  - Even if it were constructive, would be of no use since we do not know the desired function anyway, our goal is to learn it through the samples
  - But this result does give us confidence that we are on the right track
    - MNN is general enough to construct the correct decision boundaries, unlike the Perceptron

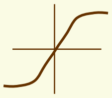
MNN Activation function

- could use a discontinuous activation function

$$f(net_k) = \begin{cases} 1 & \text{if } net_k \geq 0 \\ -1 & \text{if } net_k < 0 \end{cases}$$

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

sigmoid function



- From now on, assume ***f*** is a differentiable function

MNN: Modes of Operation

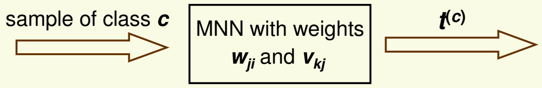
- Network have two modes of operation:
  - ***Feedforward***
    The feedforward operations consists of presenting a pattern to the input units and passing (or feeding) the signals through the network in order to get outputs units (no cycles!)
  - ***Learning***
    The supervised learning consists of 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 $\boldsymbol{x_1}$,…, $\boldsymbol{x_n}$ each of class $\boldsymbol{1}$,…,$\boldsymbol{m}$
- Let network output $\boldsymbol{z}$ represent class $\boldsymbol{c}$ as ***target*** $\boldsymbol{t^{(c)}}$

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_c \\ \vdots \\ z_m \end{bmatrix} = t^{(c)} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow c\text{th row}$$

Our Ultimate Goal For FeedForward Operation



MNN training to achieve the Ultimate Goal

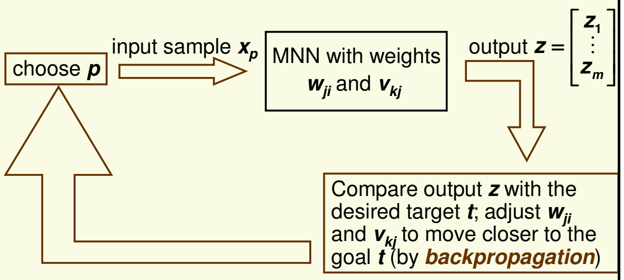
Modify (learn) MNN parameters $\boldsymbol{w_{ji}}$ and $\boldsymbol{v_{kj}}$ so that for each ***training*** sample of class $\boldsymbol{c}$ MNN output $\boldsymbol{z} = \boldsymbol{t^{(c)}}$

MNN

- Can vary
  - number of hidden layers
  - Nonlinear activation function
    - Can use different function for hidden and output layers
    - Can use different function at each hidden and output node

Network Training (learning)

1. Initialize weights $\boldsymbol{w_{ji}}$ and $\boldsymbol{v_{kj}}$ randomly **but not to 0**
2. Iterate until a stopping criterion is reached



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: $J(w,v)=\frac{1}{2}\sum_{i=1}^{n}\sum_{c=1}^{m}\left(t_c^{(i)}-z_c^{(i)}\right)^2$
- Use gradient descent:

$$\boxed{\begin{array}{l} v^{(0)},w^{(0)} = \text{random} \\ \textit{repeat until convergence:} \\ \quad w^{(t+1)}=w^{(t)}-\eta\nabla_w J\left(w^{(t)}\right) \\ \quad v^{(t+1)}=v^{(t)}-\eta\nabla_v J\left(v^{(t)}\right) \end{array}}$$

BackPropagation: Layered Model

activation at hidden unit ***j***: $net_j=\sum_{i=1}^{d}x^{(i)}w_{ji}+w_{j0}$

$\Downarrow$

output at hidden unit ***j***: $y_j=f\left(net_j\right)$

$\Downarrow$

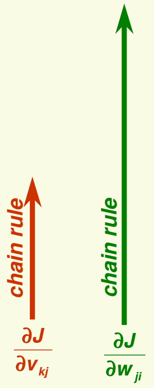
activation at output unit ***k***: $net_k^*=\sum_{j=1}^{N_H}y_jv_{kj}+v_{k0}$

$\Downarrow$

activation at output unit ***k***: $z_k=f\left(net_k^*\right)$

$\Downarrow$

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



BackPropagation

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

$$J(w,v)=\frac{1}{2}\sum_{c=1}^{m}(t_c-\underset{\text{fixed constant}}{\underbrace{z_c}^{\text{function of w,v}}})^2$$

function of w,v

fixed constant

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

- 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 $\frac{\partial J}{\partial w_{ji}}$

BackPropagation

$$net_k^*=\sum_{j=1}^{N_H}y_jv_{kj}+v_{k0} \Rightarrow z_k=f\left(net_k^*\right) \Rightarrow J(w,v)=\frac{1}{2}\sum_{c=1}^{m}(t_c-z_c)^2$$

- First compute hidden-to-output derivatives $\frac{\partial J}{\partial v_{kj}}$

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

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

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

$$=\boxed{\begin{cases}-(t_k-z_k)f'\left(net_k^*\right)y_j & \text{if } j\neq 0\\ -(t_k-z_k)f'\left(net_k^*\right) & \text{if } j=0\end{cases}}$$

BackPropagation

Gradient Descent *Single Sample* Update Rule for
hidden-to-output weights v_{kj}

j > 0: $v_{kj}^{(t+1)} = v_{kj}^{(t)} + \eta(t_k - z_k) f'(net_k^*) y_j$

j = 0 (bias weight): $v_{k0}^{(t+1)} = v_{k0}^{(t)} + \eta(t_k - z_k) f'(net_k^*)$

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: $w_{ji}^{(t+1)} = w_{ji}^{(t)} + \eta f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*) v_{kj}$

i = 0 (bias weight): $w_{j0}^{(t+1)} = w_{j0}^{(t)} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*) v_{kj}$

BackPropagation

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

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

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

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

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

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

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

$$\Downarrow$$

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

$$\Downarrow$$

$$y_j = f(net_j)$$

$$\Downarrow$$

$$net_k^* = \sum_{s=1}^{N_H} y_s v_{ks} + v_{k0}$$

$$\Downarrow$$

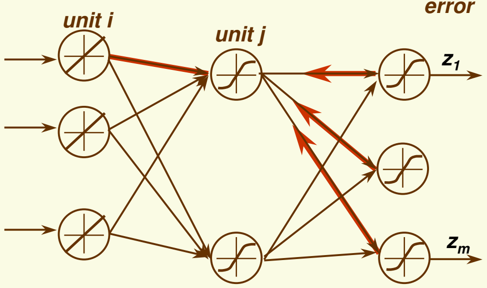
$$z_k = f(net_k^*)$$

$$\Downarrow$$

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

BackPropagation of Errors

$$\frac{\partial J}{\partial w_{ji}} = -f'(net_j) x^{(i)} \sum_{k=1}^{m} (t_k - z_k)\, f'(net_k^*) v_{kj} \qquad \frac{\partial J}{\partial v_{kj}} = -\underbrace{(t_k - z_k)}_{error} f'(net_k^*) y_j$$



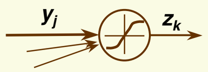
- Name “backpropagation” because during training, errors propagated back from output to hidden layer

BackPropagation

- 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 $\eta(t_k - z_k)f'(net_k^*)y_j < 0$

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

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 ***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

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  $0 \le i \le d,\ 0 \le j \le n_H,\ 0 \le k \le m$
            $v_{kj} = v_{kj} + \eta(t_k - z_k)f'(net_k^*)y_j$
            $v_{k0} = v_{k0} + \eta(t_k - z_k)f'(net_k^*)$
            $w_{ji} = w_{ji} + \eta f'(net_j)x^{(i)}\sum_{k\equiv1}^{m}(t_k - z_k)\, f'(net_k^*)v_{kj}$
            $w_{j0} = w_{j0} + \eta f'(net_j)\sum_{k=1}^{m}(t_k - z_k)\, f'(net_k^*)v_{kj}$

      $t = t + 1$
    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:

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

- Derivative of full objective function is just a sum of derivatives for each sample:

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

already derived this

Batch Back Propagation

```
1.  Initialize $n_H$, $w$, $v$, $\theta$, $\eta$, $t = 0$
2.  do
        $\Delta v_{kj} = \Delta v_{k0} = \Delta w_{ji} = \Delta w_{j0} = 0$
        one epoch:
        for all $1 \le p \le n$
            for all $0 \le i \le d,\ 0 \le j \le n_H,\ 0 \le k \le m$
                $\Delta v_{kj} = \Delta v_{kj} + \eta(t_k - z_k) f'(net_k^*) y_j$
                $\Delta v_{k0} = \Delta v_{k0} + \eta(t_k - z_k) f'(net_k^*)$
                $\Delta w_{ji} = \Delta w_{ji} + \eta f'(net_j) x_p^{(i)} \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$
                $\Delta w_{j0} = \Delta w_{j0} + \eta f'(net_j) \sum_{k=1}^{m} (t_k - z_k) f'(net_k^*) v_{kj}$
        $v_{kj} = v_{kj} + \Delta v_{kj};\ v_{k0} = v_{k0} + \Delta v_{k0};\ w_{ji} = w_{ji} + \Delta w_{ji};\ w_{j0} = w_{j0} + \Delta w_{j0}$
        $t = t + 1$
    until $\|J\| < \theta$
3.  return $v$, $w$
```

Batch Back Propagation

- For example,

$$\frac{\partial J}{\partial w_{ji}}=\sum_{p=1}^{n}-f'(net_j)\,x_p^{(i)}\sum_{k=1}^{m}(t_k-z_k)\,f'(net_k^*)\,v_{kj}$$

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