Sigmoid units, multilayer networks, Backpropagation

Relevant Readings: Section 4.5 in Mitchell

CS495 - Machine Learning, Fall 2009

Sigmoid units

- So far we have seen two kinds of units for ANNs:

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$
  - Let x_i be the inputs

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$
  - Let x_i be the inputs
  - Let w_i be the weights of the inputs

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$
  - Let x_i be the inputs
  - Let w_i be the weights of the inputs
  - If we let $y = \sum_i w_i x_i$ then the output o of a sigmoid unit is given by $\sigma(y) = 1/(1 + e^{-y})$

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$
  - Let x_i be the inputs
  - Let w_i be the weights of the inputs
  - If we let $y = \sum_i w_i x_i$ then the output o of a sigmoid unit is given by $\sigma(y) = 1/(1 + e^{-y})$
  - The neat part is that the derivative can be expressed as: $\sigma(y)' = \sigma(y) \cdot (1 - \sigma(y))$

Sigmoid units

- So far we have seen two kinds of units for ANNs:
  - Perceptrons
  - Linear units
- Both of these have deficits when it comes to building multilayer networks
  - Perceptrons are not differentiable, thus gradient descent doesn't work
  - Multilayer networks of linear units only express linear functions, thus are no better than a single layer
  - *Sigmoid units* overcome both of these difficulties
- Sigmoid units take on values in the range $(0, 1)$
  - Let x_i be the inputs
  - Let w_i be the weights of the inputs
  - If we let $y = \sum_i w_i x_i$ then the output o of a sigmoid unit is given by $\sigma(y) = 1/(1 + e^{-y})$
  - The neat part is that the derivative can be expressed as: $\sigma(y)' = \sigma(y) \cdot (1 - \sigma(y))$

Multilayer units

- We will study feedforward multilayer units

Multilayer units

- We will study feedforward multilayer units
  - The *input units* are simply the input values fed into the network

Multilayer units

- We will study feedforward multilayer units
  - The *input units* are simply the input values fed into the network
  - Each layer of *hidden units* are sigmoid units, taking inputs from the previous layer

Multilayer units

- We will study feedforward multilayer units
  - The *input units* are simply the input values fed into the network
  - Each layer of *hidden units* are sigmoid units, taking inputs from the previous layer
  - The *output units* (perhaps sigmoid, perhaps not) take input from the previous layer and give the output of the network

Multilayer units

- We will study feedforward multilayer units
  - The *input units* are simply the input values fed into the network
  - Each layer of *hidden units* are sigmoid units, taking inputs from the previous layer
  - The *output units* (perhaps sigmoid, perhaps not) take input from the previous layer and give the output of the network

Backpropagation

- Start with small random weights on the edges

Backpropagation

- Start with small random weights on the edges
- The basic idea:

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output
  - The error of each hidden unit is recursively defined to be the weighted sum error of its "downstream units" times $o(1-o)$ where o is the unit's output

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output
  - The error of each hidden unit is recursively defined to be the weighted sum error of its "downstream units" times $o(1-o)$ where o is the unit's output
  - Every unit updates the weights w_i of its inputs by adding $\eta\delta x_i$ where η is a small constant, δ is the error of this unit, and x_i are the inputs to this unit

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output
  - The error of each hidden unit is recursively defined to be the weighted sum error of its "downstream units" times $o(1-o)$ where o is the unit's output
  - Every unit updates the weights w_i of its inputs by adding $\eta\delta x_i$ where η is a small constant, δ is the error of this unit, and x_i are the inputs to this unit
  - Repeat over all the training instances, many times

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output
  - The error of each hidden unit is recursively defined to be the weighted sum error of its "downstream units" times $o(1-o)$ where o is the unit's output
  - Every unit updates the weights w_i of its inputs by adding $\eta\delta x_i$ where η is a small constant, δ is the error of this unit, and x_i are the inputs to this unit
  - Repeat over all the training instances, many times
- Full pseudocode in Table 4.2 in Mitchell

Backpropagation

- Start with small random weights on the edges
- The basic idea:
  - Evaluate the network on the training instance to get the network output values
  - The error of each output unit is defined to be $o(1-o)(t-o)$ where o is the unit output and t is the training output
  - The error of each hidden unit is recursively defined to be the weighted sum error of its "downstream units" times $o(1-o)$ where o is the unit's output
  - Every unit updates the weights w_i of its inputs by adding $\eta\delta x_i$ where η is a small constant, δ is the error of this unit, and x_i are the inputs to this unit
  - Repeat over all the training instances, many times
- Full pseudocode in Table 4.2 in Mitchell

Backpropagation

- The Backpropagation algorithm seeks to minimize the squared error of the network outputs, summed over all training examples

Backpropagation

- The Backpropagation algorithm seeks to minimize the squared error of the network outputs, summed over all training examples
- Since the error surface is more complicated, the algorithm could get stuck in a local minimum

Backpropagation

- The Backpropagation algorithm seeks to minimize the squared error of the network outputs, summed over all training examples
- Since the error surface is more complicated, the algorithm could get stuck in a local minimum
  - To avoid this, one can train several ANNs and the best one is kept

Backpropagation

- The Backpropagation algorithm seeks to minimize the squared error of the network outputs, summed over all training examples
- Since the error surface is more complicated, the algorithm could get stuck in a local minimum
  - To avoid this, one can train several ANNs and the best one is kept
- Choose a stopping criterion wisely (don't overtrain, as this leads to poor generalization)

Backpropagation

- The Backpropagation algorithm seeks to minimize the squared error of the network outputs, summed over all training examples
- Since the error surface is more complicated, the algorithm could get stuck in a local minimum
  - To avoid this, one can train several ANNs and the best one is kept
- Choose a stopping criterion wisely (don't overtrain, as this leads to poor generalization)