Automated Stock Trading using Neural Networks

Matthew Knowles

Department of Mathematics
University of York

mk1320@york.ac.uk

November 22, 2021

Mathematical Underpinning of Neural Networks

Mathematical Underpinning of Neural Networks

- Layers of nodes (perceptrons). k "hidden" layers sandwiched by an input layer and an output layer.

Mathematical Underpinning of Neural Networks

- Layers of nodes (perceptrons). k "hidden" layers sandwiched by an input layer and an output layer.
- Each node in layer j connects to each node in layer $j + 1$. Connection is defined by a weight and a bias.

Mathematical Underpinning of Neural Networks

- Layers of nodes (perceptrons). k “hidden” layers sandwiched by an input layer and an output layer.
- Each node in layer j connects to each node in layer $j+1$. Connection is defined by a weight and a bias.
- Let the layer k_i have m nodes, and the layer k_i+1 have n nodes. Then the weights between the layers are given by the matrix

Mathematical Underpinning of Neural Networks

- Layers of nodes (perceptrons). k “hidden” layers sandwiched by an input layer and an output layer.
- Each node in layer j connects to each node in layer $j + 1$. Connection is defined by a weight and a bias.
- Let the layer k_i have m nodes, and the layer $k_i + 1$ have n nodes. Then the weights between the layers are given by the matrix

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{bmatrix} \tag{1}$$

Mathematical Underpinning of Neural Networks

- Layers of nodes (perceptrons). k "hidden" layers sandwiched by an input layer and an output layer.
- Each node in layer j connects to each node in layer $j+1$. Connection is defined by a weight and a bias.
- Let the layer k_i have m nodes, and the layer k_i+1 have n nodes. Then the weights between the layers are given by the matrix

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,m} \\ w_{2,1} & w_{2,2} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,m} \end{bmatrix} \tag{1}$$

- The values of the nodes in layer $k+1$, denoted $A_{(k+1)}$ is given by the matrix equation $A_{(k+1)} = WA_{(k)} + b_{(k)}$. Where b is the vector containing the biases.

Backpropogation

Backpropogation

- But how does this network learn? The answer: backpropogation!

Backpropogation

- But how does this network learn? The answer: backpropogation!
- Define an error function $E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i' - y_i)^2$. Where θ incorperates the weights and biases.

Backpropogation

- But how does this network learn? The answer: backpropogation!
- Define an error function $E(X, \theta) = \frac{1}{2N} \sum_{i=1}^{N} (y_i' - y_i)^2$. Where θ incorperates the weights and biases.
- The idea behind backpropogation is to find a local minimum of this function (called gradien descent). We then updated the weights and biases according to the differential of $E(X, \theta)$ with respect to each weight. For the k^{th} layer, update the weight between node i in k to j in $k + 1$, we have:

Backpropogation

- But how does this network learn? The answer: backpropogation!
- Define an error function $E(X,\theta) = \frac{1}{2N}\sum_{i=1}^{N}(y_i' - y_i)^2$. Where θ incorperates the weights and biases.
- The idea behind backpropogation is to find a local minimum of this function (called gradien descent). We then updated the weights and biases according to the differential of $E(X,\theta)$ with respect to each weight. For the k^{th} layer, update the weight between node i in k to j in $k+1$, we have:

$$\Delta w_{ij}^k = -\alpha \frac{\partial E(X,\theta)}{\partial w_{ij}^k} \tag{2}$$

. Where α is called the *learning rate*.

Esketit

Oooh, lil pump