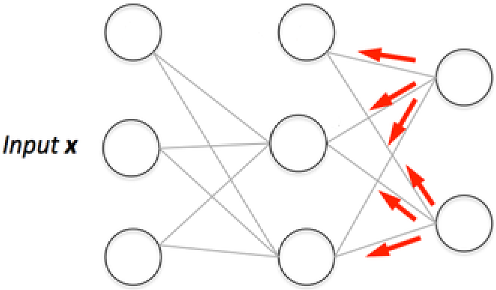
Artificial Intelligence

ANN – Back propagation

Jonathan Mwaura

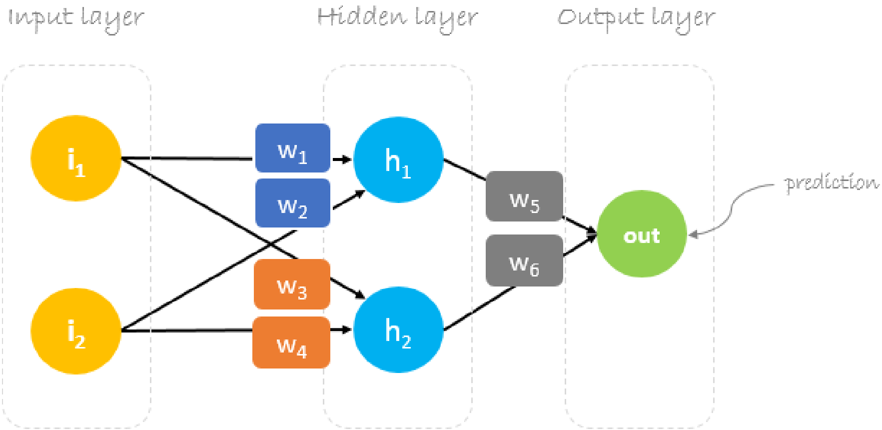
December 6, 2018

Backpropagation Step by Step

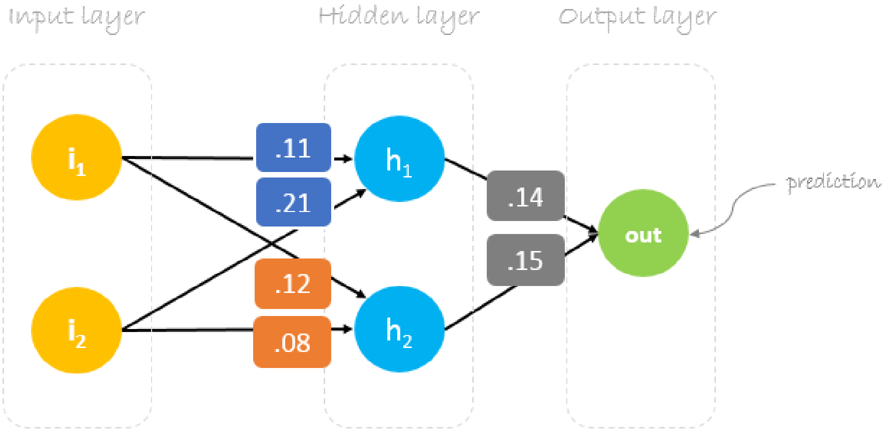


Backpropagation is the technique used to train an ANN.

Example overview



Weights, Weights, Weights



What exactly is NN Training?

Neural network training is about finding weights that minimize prediction error.

Start training with a set of randomly generated weights.

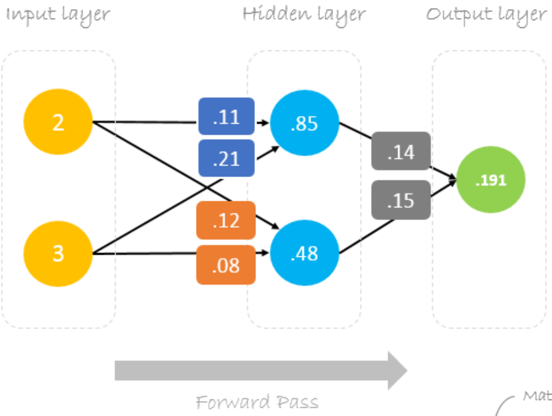
Then, use backpropagation to update the weights in an attempt to correctly map arbitrary inputs to outputs.

Dataset

Our dataset has one sample with two inputs and one output.

Our single sample is as following inputs=[2, 3] and output=[1].

Forward pass



Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.11 & 0.12 \\ 0.21 & 0.08 \end{bmatrix} = [0.85 \quad 0.48] \cdot \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} = [0.191]$$

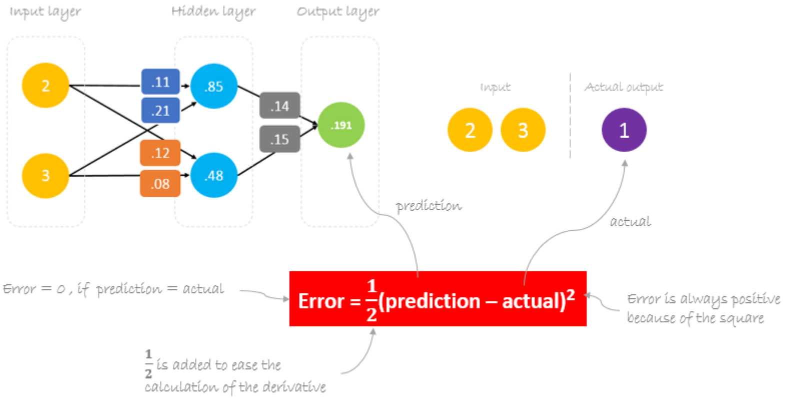
Details

2 x .11 + 3 x .21 = .85

.85 x .14 + .48 x .15 = .191

2 x .12 + 3 x .08 = .48

Calculate the Error



$$\text{Error} = \frac{1}{2}(0.191 - 1.0)^2 = 0.327$$

Reducing the error

$$\textbf{prediction} = out$$

$$\textbf{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$
$$h_2 = i_1 w_3 + i_2 w_4$$

$$\textbf{prediction} = (i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6$$

to change ***prediction*** value,
we need to change ***weights***

The question now is how to change (update) the weights value so that the error is reduced? The answer is Backpropagation!

Reducing the error

The goal of the training is to reduce the error or the difference between prediction and actual output.

Actual output is constant, “not changing”, the only way to reduce the error is to change prediction value.

Decomposing prediction into its basic elements we can find that weights are the variable elements affecting prediction value.

In other words, in order to change prediction value, we need to change weights values.

So..Backpropagation..

Backpropagation, short for "backward propagation of errors" is a mechanism used to update the weights using gradient descent.

BP computes the gradient of the error function with respect to the neural network's weights.

The calculation proceeds backwards through the network.

Gradient Descent

an iterative optimization algorithm for finding the minimum of a function; in our case we want to minimize th error function. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

Weight Update Formula

Derivative of Error with respect to weight

Old weight

$$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$$

New weight

Learning rate

Update Example

$$^{*}W_6 = W_6 - \mathbf{a}\left(\frac{\partial Error}{\partial W_6}\right)$$

Optionally, we multiply the derivative of the error function by a selected number to make sure that the new updated weight is minimizing the error function; this is called **learning rate.**

Chain Rule

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1$ w

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\, (i_1\, w_1 + i_2\, w_2)\, w_5 + (i_1\, w_3 + i_2\, w_4)\, w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\text{predictoin} - \text{actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (i_1\, w_3 + i_2\, w_4)$$

$h_2 = i_1\, w_3 + i_2\, w_4$

$$\frac{\partial Error}{\partial W_6} = (\text{predictoin} - \text{actula}) * (h_2)$$

Δ = prediction – actual

delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Update

$$^{*}W_6 = W_6 - a\,\Delta h_2$$

Similarly, we can derive the update formula for w5 and any other weights existing between the output and the hidden layer.

$$^{*}W_5 = W_5 - a\,\Delta h_1$$

Weights associated with Hidden Layer

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ chain rule

$$\text{Error} = \frac{1}{2}(\text{prediction} - \text{actual})^2$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\, (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin} - \mathbf{actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin} - \mathbf{actula}) * (w_5 i_1)$$

$$\Delta = \text{prediction} - \text{actual}$$ delta

$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Moving backward to update w1, w2, w3 and w4 existing between input and hidden layer, the partial derivative for the error function with respect to w1, for example,

Weights associated with Hidden Layer

$$
\begin{aligned}
{}^*w_6 &= w_6 - a\,(h_2 \,.\, \Delta) \\
{}^*w_5 &= w_5 - a\,(h_1 \,.\, \Delta) \\
{}^*w_4 &= w_4 - a\,(i_2 \,.\, \Delta w_6) \\
{}^*w_3 &= w_3 - a\,(i_1 \,.\, \Delta w_6) \\
{}^*w_2 &= w_2 - a\,(i_2 \,.\, \Delta w_5) \\
{}^*w_1 &= w_1 - a\,(i_1 \,.\, \Delta w_5)
\end{aligned}
$$

updated weights

We can rewrite the update formulas in matrices as following

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} ah_1\Delta \\ ah_2\Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1\Delta w_5 & a\, i_1\Delta w_6 \\ a\, i_2\Delta w_5 & a\, i_2\Delta w_6 \end{bmatrix}$$

Backward Pass

Using derived formulas we can find the new weights.

L

earning rate: is a hyperparameter which means that we need to manually guess its value.

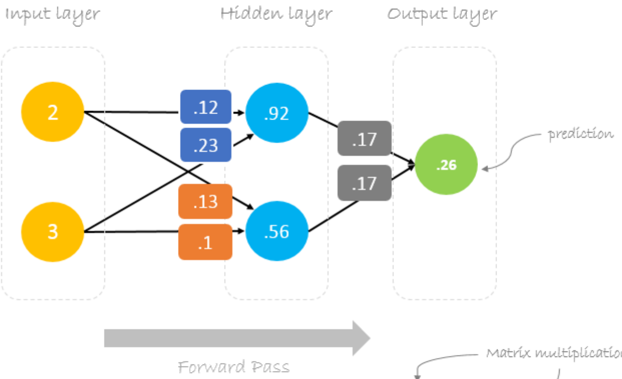
$\Delta = 0.191 - 1 = -0.809$ Delta = prediction - actual

$a = 0.05$ Learning rate, we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . \begin{bmatrix} 0.14 & 0.15 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Backward Pass



Matrix multiplication

$$[2 \quad 3] \cdot \begin{bmatrix} 0.12 & 0.13 \\ 0.23 & 0.10 \end{bmatrix} = [0.92 \quad 0.56] \cdot \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix} = [0.26]$$

Details

2 x .12 + 3 x .23 = .85

.92 x .17 + .56 x .17 = .26

2 x .13 + 3 x .10 = .48

Consider a system with two states and two actions. You perform actions and observe the rewards and transitions listed below. Each step lists the current state, reward, action and resulting transition as S_i; $R = r$; a_k: $S_i \longrightarrow S_j$. Perform Q-learning using a learning rate of $\alpha = 0.5$ and a discount factor of $\gamma = 0.5$ for each step. The Q-table entries are initialized to zero.

Q	S_1	S_2
a_1	0	0
a_2	0	0

Table: S_1; R = -10; a_1: $S_1 \longrightarrow S_2$