Machine Learning

Backpropagation 2

Mostafa S. Ibrahim
Teaching, Training and Coaching for more than a decade!

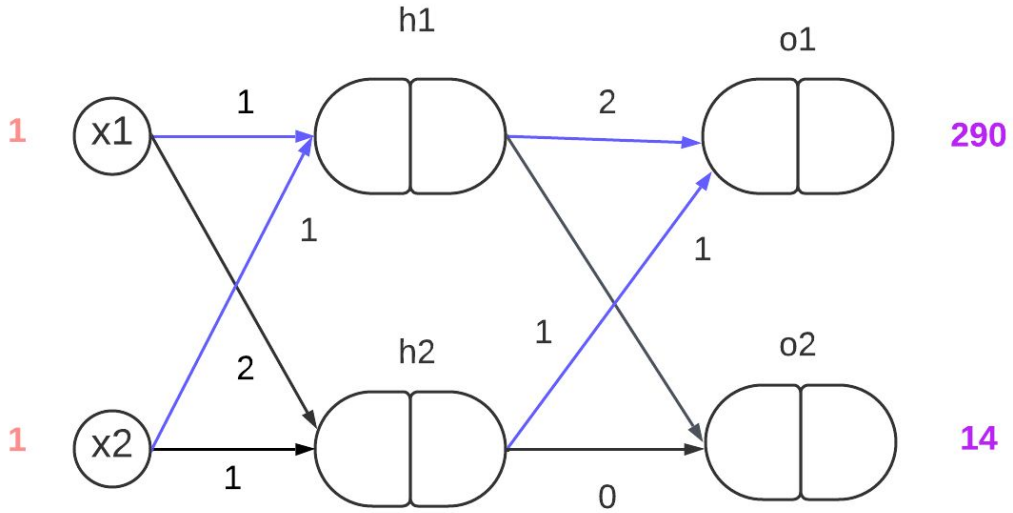
Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / MSc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

Note

- Typically to train a network we initialize with values around zero
- We use activation function such as sigmoid, tanh and reul
- However, this makes tracing NN for educational examples a bit hard
- To simplify the process, I will be using the following techniques:
  - Use simple network
  - Use integer weights
  - Use activation function: f(net) = net^2, which has simple derivative 2net
    - Hence all computations are integers

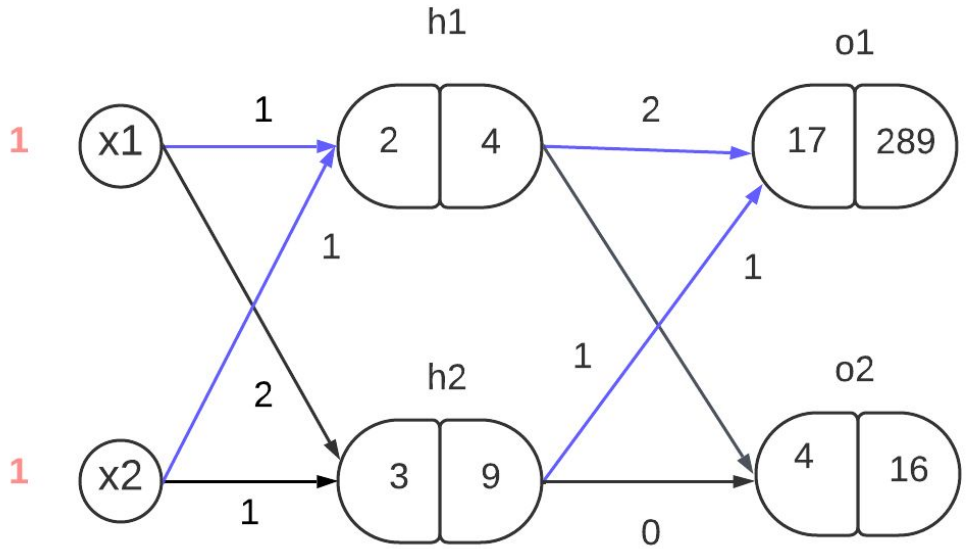
Your turn: Calc Feedforward Results

- Compute the feedforward values for this node
  - Assume inputs are (1, 1) and target outputs are (290, 14)
- Assume both hidden and output layers have activation function f(net) = net^2
- Assume the error is based on MSE = ½ (output - target)^2



Feedforward

- First cell sums 1*1 + 1*1 = 2. Then f(2) = 2^2 = 4
- Second cell sums 2*1 + 1*1 = 3. Then f(3) = 3^2 = 9
- 17 = 2*4 + 1 * 9. 17 * 17 = 289



290 => E1 = 1/2 (289-290)^2

14 => E2 = 1/2 (16-14)^2

Backpropagation

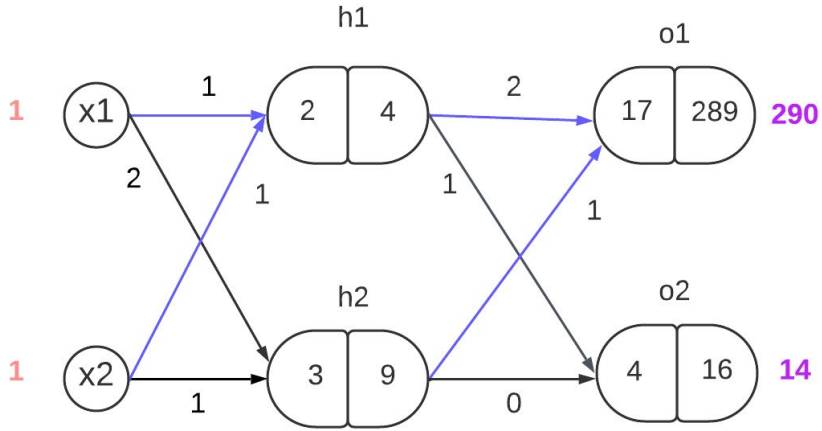
- First, perform the feedforward of an example (or minibatch)
- Compute derivatives of nodes (net/out)
  - Cache the results of node_net
  - Do that for all nodes first
- Compute derivatives of the weights
  - We better do that AFTER the last stage to make sure derivatives form other nodes are based on old weights

Recall the Formulas

- ∂E/∂d_out = ∑ w[i] x ∂E/∂n[i]_net (∂n[i]_net cached)
- ∂E/∂d_net = ∂E/∂d_out * ∂d_out/∂d_net
- ∂E/∂w1 = ∂E/∂d_net * ∂d_net/∂w1
- Starting from the output node with MSE loss
  - ∂E/∂o_out = output - target

Compute ∂E/∂o_out

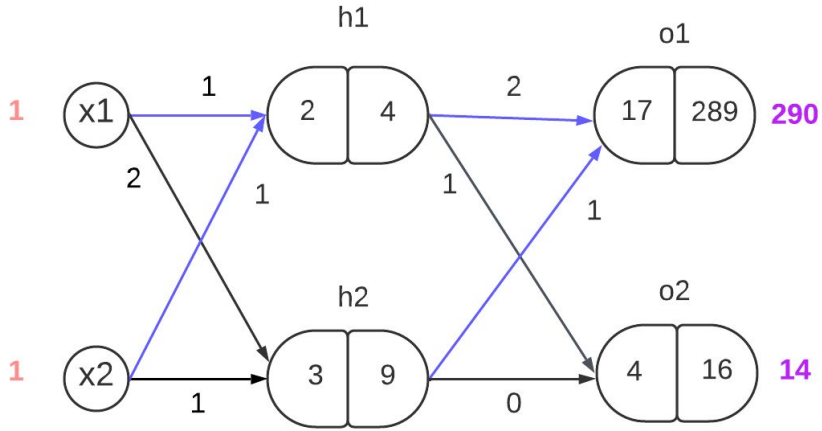
- This is the base case, which is simple: **output-target for MSE**
- ∂E/∂o1_out = 289 - 290 = -1
- ∂E/∂o2_out = 16 - 14 = 2



Compute ∂E/∂o_net

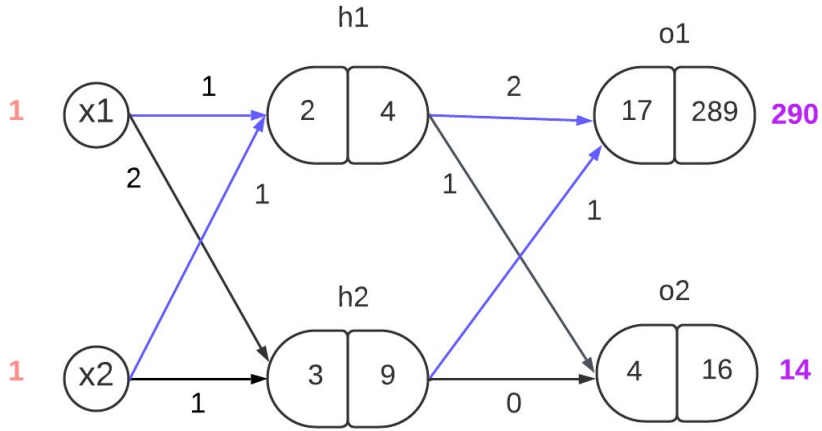
- Use formula: ∂E/∂o_net = ∂E/∂o_out * ∂o_out/∂o_net
  - We just computed ∂E/∂o_out {-1, 2}
  - ∂o_out/∂o_net = derivative of net^2 = 2 x net. Net values are {17, 4}

- ∂E/∂o1_net = -1 * 2 * 17 = -34
- ∂E/∂o2_net = 2 * 2 * 4 = 16
- **Cache** these values!
  - Literature call them **delta**
  - Used symbol: Δ or more common δ



Compute ∂E/∂h_out

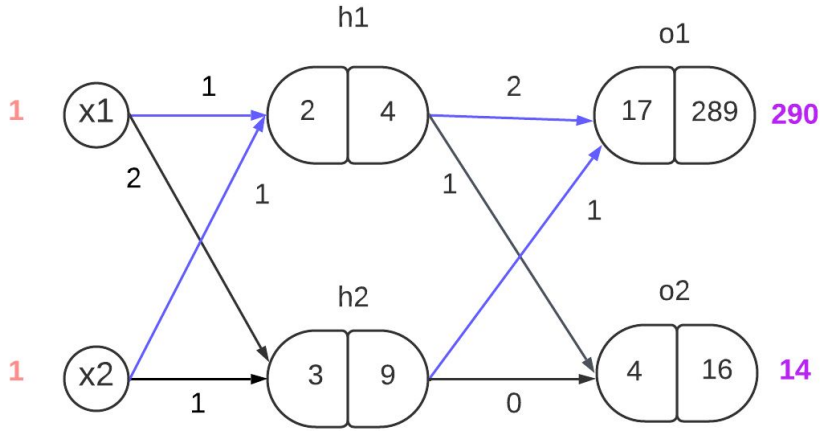
- Use formula: ∂E/∂h_out = ∑ w[i] x ∂E/∂n[i]_net
  - From cache: ∂E/∂n[i]_net: {-34, 16}
- ∂E/∂h1_out = 2 * -34 + 1 * 16 = -52
- ∂E/∂h2_out = 1 * -34 + 0 * 16 = -34



Compute ∂E/∂h_net

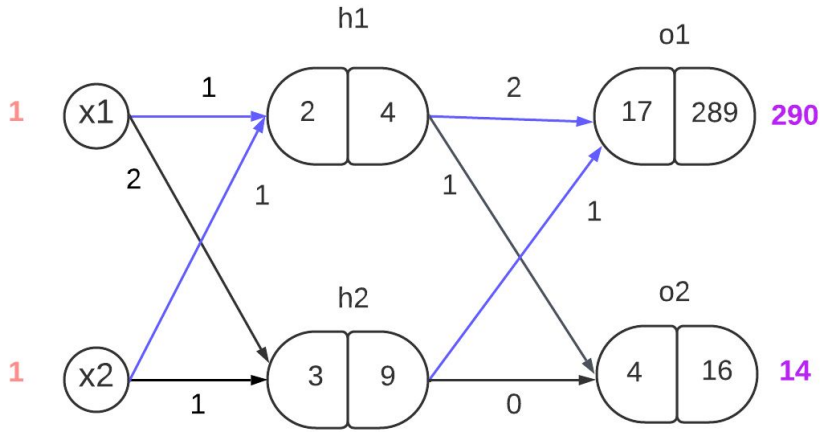
- Use formula: ∂E/∂h_net = ∂E/∂h_out * ∂h_out/∂h_net
  - We just computed ∂E/∂h_out {-52, -34}
  - ∂h_out/∂h_net = derivative of net^2 = 2 x net {2, 3}

- ∂E/∂h1_net = -52 * 2 * 2 = -208
- ∂E/∂h2_net = -34 * 2 * 3 = -204



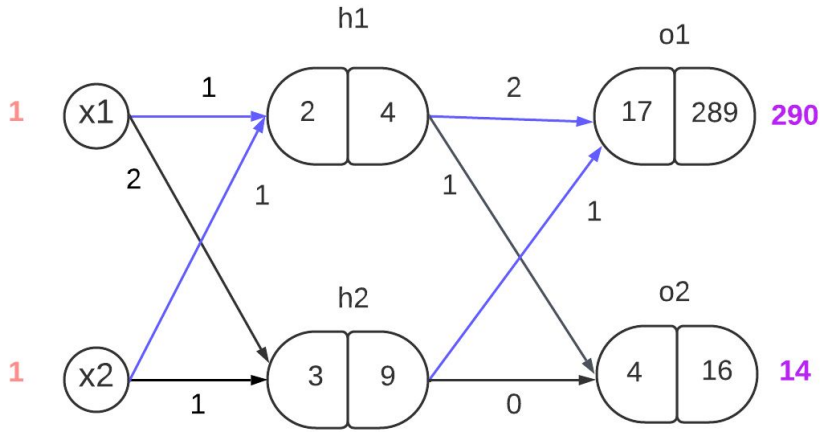
Updating weights

- Now this is the trivial part
- First compute the derivative using: ∂E/∂w = ∂E/∂d_net * ∂d_net/∂w
- Example edge (h1 - o1), which has w = 2
  - ∂E/∂o1_net = -34
  - ∂o1_net/∂w = h1_out = 4
  - ∂E/∂w = -34 * 4 = -136
- Updating the weight
  - Assume alpha = 0.5
  - new_w = 2 - 0.5 * -136 = 70



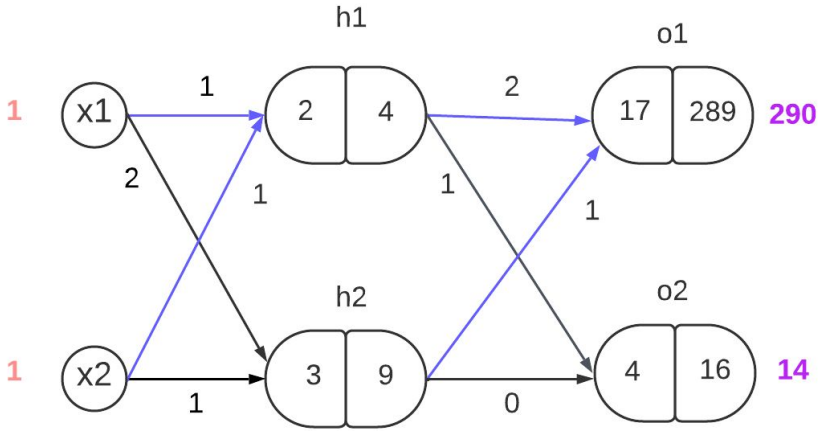
Updating weights

-
- First compute the derivative using: ∂E/∂w = ∂E/∂d_net * ∂d_net/∂w
- Example edge (x1 - h2), which has w = 1
  - ∂E/∂h2_net = -204
  - ∂h2_net/∂w = x2 = 1
  - ∂E/∂w = -204 * 1 = -204
- Second, updating the weight as in gradient descent
  - Assume the learning rate = 0.5
  - new_w = old_w - lr x ∂E/∂w
  - new_w = 1 - 0.5 * -204 = 103



Your turn to validate

- Edge: h2-o2, ∂E/∂w = 144
- Edge: x1-h1, ∂E/∂w = -208



Notes

- In regression tasks, it is common to have no activation function on the output nodes
- We used activation function out = f(net) = net^2, which has derivative 2net
- If we used sigmoid out = sigmoid(net), the derivative would be out * (1-out)
- We can add a common bias per layer and just follows the same rules
- Recall Linear Regression cost function with MSE was convex. With NN nonlinear activation functions, the cost function is non-convex
- NN with linear activation functions in all neurons can be reduced to the standard input-output NN (linear regression)

repeat

for each weight $w_{i,j}$ in the network **do**

$w_{i,j} \leftarrow$ a small random number

for each example (x, y) **do**

/*Propagate the inputs forward to compute the outputs*/

for each node i in the input layer **do**

$a_i \leftarrow x_i$

for $\ell = 2$ **to** L **do**

for each node j in layer ℓ **do**

$v_j \leftarrow \sum_i w_{i,j} a_i$

$a_j \leftarrow h(v_j)$

/*Propagate the deltas backwards from output layer to input layer*/

for each node j in output layer **do**

$\Delta[j] \leftarrow h'(v_j)(y_j - a_j)$

for $\ell = L - 1$ **to** 1 **do**

for each node i in layer ℓ **do**

$\Delta[i] \leftarrow h'(v_i) \sum_j w_{i,j} \Delta[j]$

/*Update every weight in network using deltas*/

for each weight $w_{i,j}$ in network **do**

$w_{i,j} \leftarrow w_{i,j} + \alpha a_i \Delta[j]$

until some stopping criterion is satisfied

src

About Neural Network

- Neural networks are powerful non-linear learning machines
- Neural networks are subject to getting stuck in local minima
- Larger networks and datasets may take longer to train (convergence)
- Deep networks can achieve remarkable performance across a wide range of problems, especially on semi-structured data (text, image, audio)

Universal approximation theorem

- A feedforward network can **approximate any continuous** function, even with a **single** hidden layer
  - This is a very interesting fact that other ML techniques lack
  - Mathematical proof is not easy
- In **practice** however, NN performs close to many other ML algorithms.
  - We don’t know how to **train** to a **very huge** NN to be a **universal** approximator
  - There are external **limitations** that limits this theory in practice
    - Limitations of enough data availability and proper architectures
    - Errors from data and optimizers
- Since 2012, Deep learning community found setup to build **deep networks** and techniques to train them to enlarge the scope of success, given the availability of data, compute and smart models

Relevant Materials

- Simulating examples: video, video, video
- Can neural networks solve any problem? Link Link

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."

Matrix Impl