FUNDAMENTALS OF MACHINE LEARNING IN DATA SCIENCE

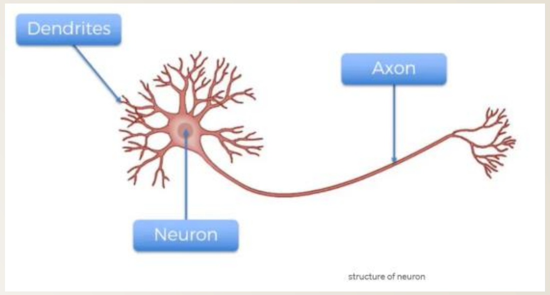
CSIS 3290

FUNDAMENTALS OF NEURAL NETWORKS

IN MACHINE LEARNING (2)

FATEMEH AHMADI

Backpropagation Algorithm

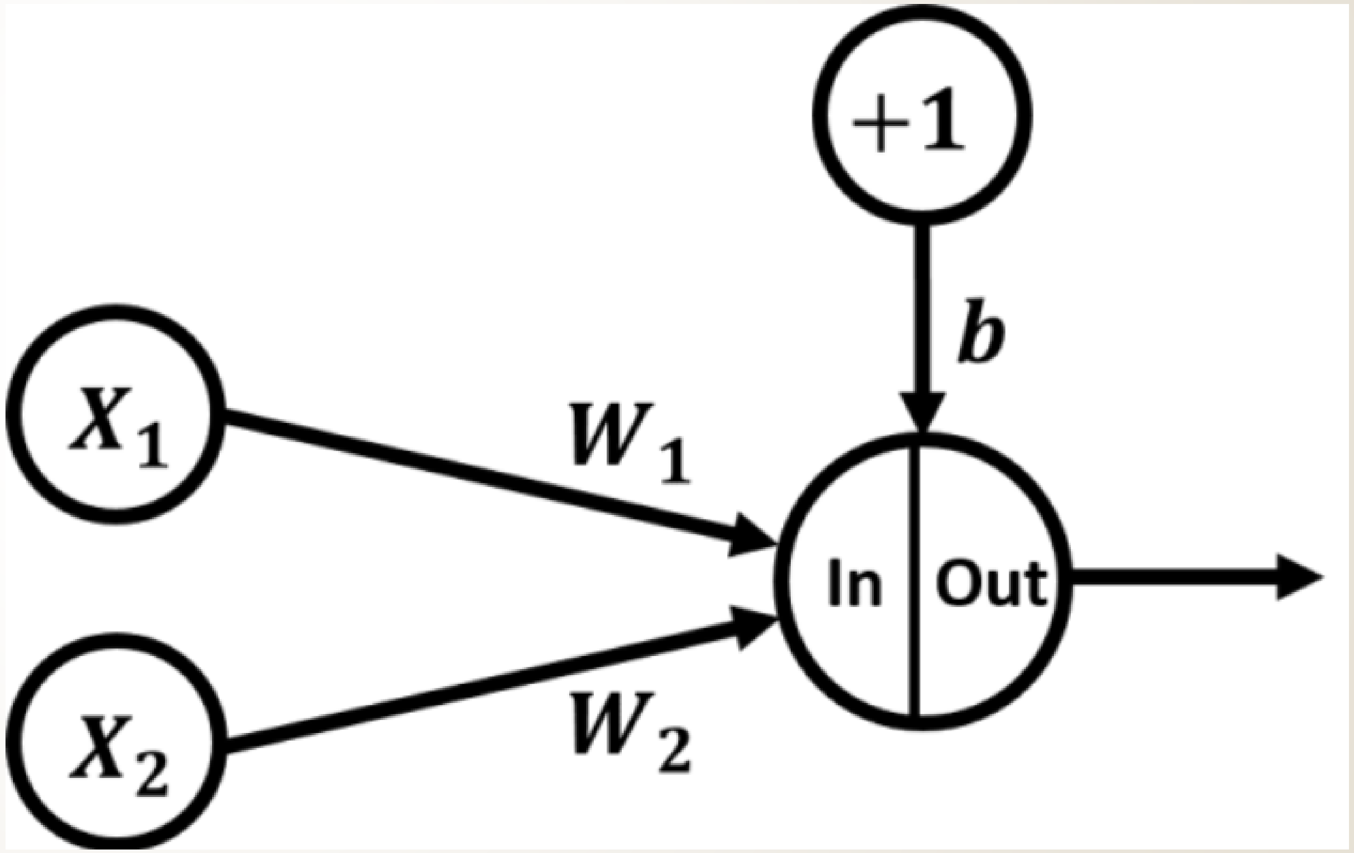


Here are some of the advantages of the backpropagation algorithm:

- It's memory-efficient in calculating the derivatives, as it uses less memory compared to other optimization algorithms, like the genetic algorithm. This is a very important feature, especially with large networks.
- The backpropagation algorithm is fast, especially for small and medium-sized networks. As more layers and neurons are added, it starts to get slower as more derivatives are calculated.
- This algorithm is generic enough to work with different network architectures, like convolutional neural networks, generative adversarial networks, fully-connected networks, and more.
- There are no parameters to tune the backpropagation algorithm, so there's less overhead. The only parameters in the process are related to the gradient descent algorithm, like learning rate.

Backpropagation Algorithm

How the algorithm works is best explained based on a simple network, like the one given in this figure. It only has an input layer with 2 inputs (X_1 and X_2), and an output layer with 1 output. There are no hidden layers.



Backpropagation Algorithm

- ✓ The weights of the inputs are W_1 and W_2, respectively.
- ✓ The bias is treated as a new input neuron to the output neuron which has a fixed value +1 and a weight b.
- ✓ Both the weights and biases could be referred to as **parameters**.
- ✓ Let's assume that output layer uses the sigmoid activation function.
- ✓ Where s is the **sum of products (SOP)** between each input and its corresponding weight:

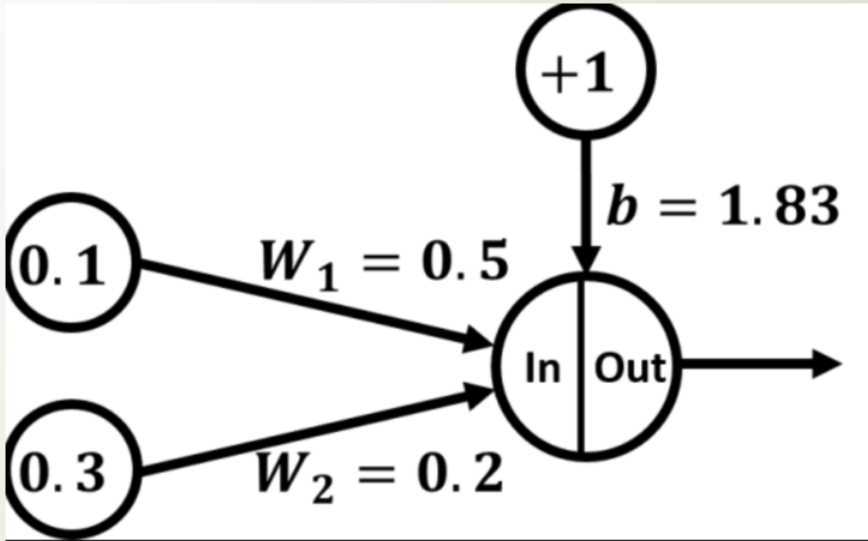
Backpropagation Algorithm

X1	X2	Desired Output
0.1	0.3	0.03

Assume that the initial values for both weights and bias are like in the next table

W1	W2	b
0.5	0.2	1.83

$$s=X_1 * W_1 + X_2 * W_2 + b$$



Forward and Backward Pass

- Now, let’s train the network and see how the network will predict the output of the sample based on the current parameters.

- As we discussed previously, the training process has 2 phases, **forward and backward**.

Forward Pass

$s=X_1* W_1+ X_2*W_2+b$

$s=0.1* 0.5+ 0.3*0.2+1.83$

$s=1.94$

The value 1.94 is then applied to the activation function (sigmoid), which results in the value 0.874352143.

$$f(s)= \frac{1}{1+e^{-x}} = \frac{1}{1+e^{-1.94}} = 0.874352143$$

Cost Function

- The output of the activation function from the output neuron reflects the predicted output of the sample. It’s obvious that there’s a difference between the desired and expected output.

- The error functions tell how close the predicted output(s) are from the desired output(s). The optimal value for error is **zero**, meaning there’s no error at all, and both desired and predicted results are identical. One of the error functions is the **squared error function**, as defined in the next equation:

$$E = \frac{1}{2}(desired - predicted)^2$$

Cost Function

Based on the error function, we can measure the error of our network as follows:

$$E = \frac{1}{2}(0.03 - 0.874352143)^2 = 0.356465271$$

The result shows that there is an error, and a large one: (**~0.357**). The error just gives us an indication of how far the predicted results are from the desired results.

Parameters Update Equation

The parameters can be changed according to the next equation:

$$W_{(n+1)}=W(n)+\eta[d(n)-Y(n)]X(n)$$

Where:

- n: Training step (0, 1, 2, …).
- W(n): Parameters in current training step.
- η: Learning rate with a value between 0.0 and 1.0.
- d(n): Desired output.
- Y(n): Predicted output.
- X(n): Current input at which the network made false prediction.

Parameters Update Equation

- For our network, these parameters have the following values:
- n: 0
- W(n): [1.83, 0.5, 0.2]
- η: Because it is a hyperparameter, then we can choose it 0.01 for example.
- d(n): [0.03].
- Y(n): [0.874352143].
- X(n): [+1, 0.1, 0.3]. First value (+1) is for the bias.

Parameters Update Equation

X1	X2	Desired Output
0.1	0.3	0.03

We can update our network parameters as follows:

$W_{(n+1)}=W(n)+\eta[d(n)-Y(n)]X(n)$

$=[1.83, 0.5, 0.2]+0.01[0.03-0.874352143][+1, 0.1, 0.3]$

$=[1.83, 0.5, 0.2]+0.01[-0.844352143][+1, 0.1, 0.3]$

$=[1.83, 0.5, 0.2]+-0.00844352143[+1, 0.1, 0.3]$

$=[1.83, 0.5, 0.2]+[-0.008443521, -0.000844352, -0.002533056]$

$=[1.821556479, 0.499155648, 0.197466943]$

W1	W2	b
0.5	0.2	1.83

The new parameters are listed in the next table:

W1New	W2New	bNew
0.499155648	0.197466943	1.821556479

Parameters Update Equation

For example, the backpropagation algorithm could tell us useful information, like that increasing the current value of W1 by 1.0 increases the network error by 0.07. This shows us that a smaller value for W1 is better to minimize the error.

Drawbacks of the backpropagation algorithm

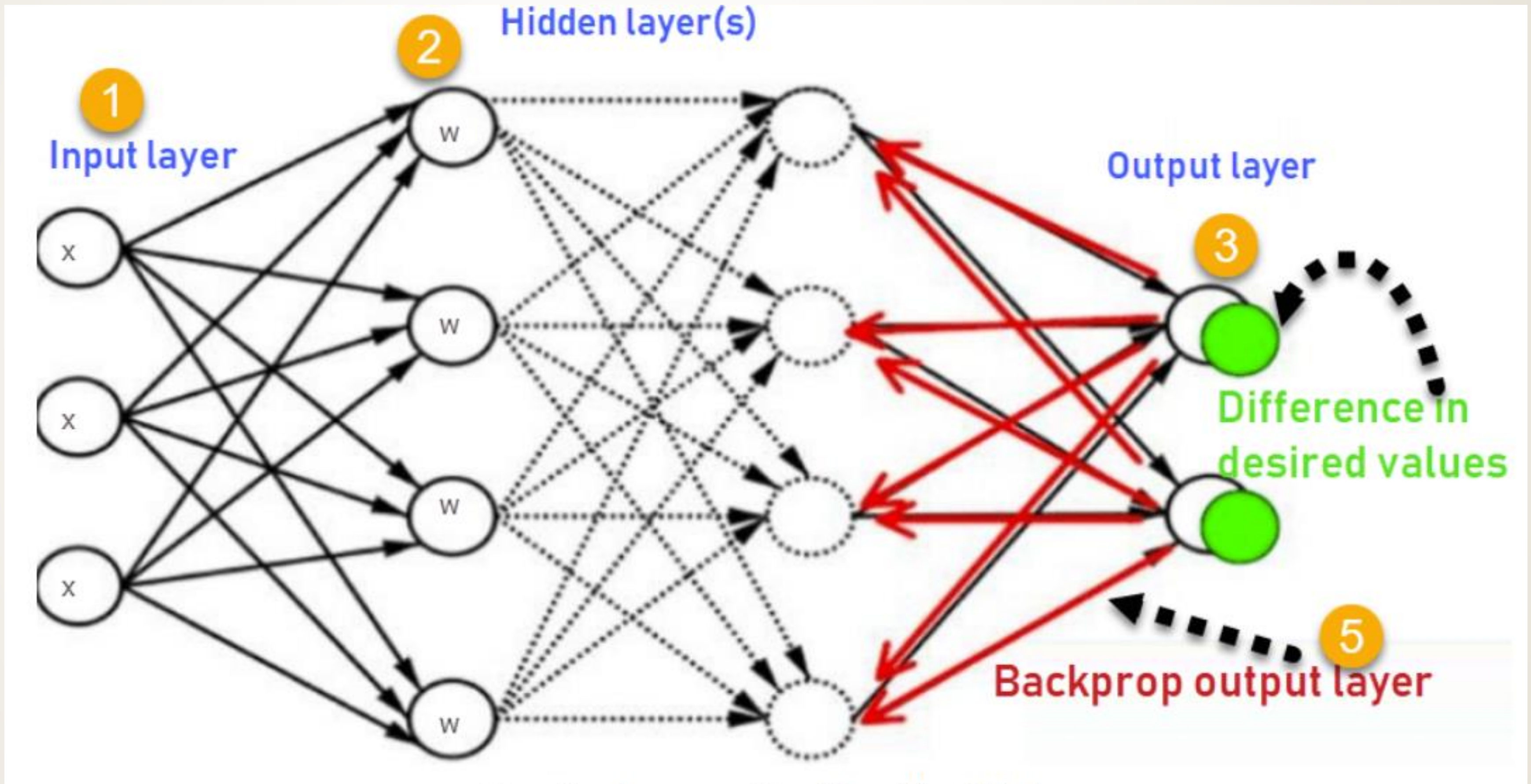
Even though the backpropagation algorithm is the most widely used algorithm for training neural networks, it has some drawbacks:

- The network should be designed carefully to avoid the vanishing and exploding gradients that affect the way the network learns. For example, the gradients calculated out of the sigmoid activation function may be very small, close to zero, which makes the network unable to update its weights. As a result, no learning happens.
- The backpropagation algorithm considers all neurons in the network equally and calculates their derivatives for each backward pass. Even when dropout layers are used, the derivatives of the dropped neurons are calculated, and then dropped.
- Backpropagation relies on infinitesimal effects (partial derivatives) to perform credit assignment. This could become a serious issue as one considers deeper and more non-linear functions.

Drawbacks of the backpropagation algorithm

- It expects that the error function is convex. For a non-convex function, backpropagation might get stuck in a local optima solution.
- The error function and the activation function must be differentiable in order for the backpropagation algorithm to work. It won't work with non-differentiable functions.
- In the forward pass, layer *i+1* must wait the calculations of layer *i* to complete. In the backward pass, layer *i* must wait layer *i+1* to complete. This makes all layers of the network locked, waiting for the remainder of the network to execute forwards and propagate error backwards, before they can be updated.

Backpropagation



How Backpropagation Algorithm Works

Reference

- ✓ https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide
- ✓ https://medium.com/swlh/fundamentals-of-neural-network-in-machine-learning-44fd9b04b825