Understanding the Forward-Forward Algorithm in Neural Networks

Tong Wu

October 17, 2024

1 The Forward-Forward Algorithm

The forward-forward algorithm is a novel approach to training neural networks, proposed as an alternative to the traditional back-propagation method. Instead of propagating errors backward, two forward passes are made: one with positive (real) samples and one with negative (synthetic or noisy) samples. The objective is to maximize a goodness function for positive samples and minimize it for negative samples, thus training the network.

1.1 Algorithm Overview

The forward-forward algorithm works as follows:

- Perform two forward passes:
  - **Positive Pass:** Forward pass with real (positive) data samples.
  - **Negative Pass:** Forward pass with synthetic or noisy (negative) samples.
- Compute a goodness score for each layer in the network.
- Adjust the weights and biases so that the goodness score for positive samples increases, while the score for negative samples decreases.

2 Example: Two-Layer Neural Network

Let's consider a simple two-layer neural network:

- Input layer with 3 neurons
- One hidden layer with 2 neurons
- Output layer with 1 neuron

The weight matrices and bias vectors for this network are as follows:

$$W^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.8 & 0.1 & 0.3 \end{bmatrix}, \quad b^{(1)} = \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} 0.3 & 0.7 \end{bmatrix}, \quad b^{(2)} = 0.2$$

The activation function $f(z)$ for all layers is the sigmoid function:

$$f(z) = \frac{1}{1 + e^{-z}}$$

2.1 Generating Positive and Negative Samples

2.1.1 Positive Samples

Let's assume we are working with real input data $x = \begin{bmatrix} 0.9 \\ 0.1 \\ 0.8 \end{bmatrix}$. This is our **positive sample**.

2.1.2 Negative Samples

For negative samples, we generate synthetic or noisy data. A simple approach is to add random noise to the input, such as using the negative sample $x_{\text{neg}} = \begin{bmatrix} 0.2 \\ 0.5 \\ 0.1 \end{bmatrix}$. This noisy input acts as the **negative sample**.

3 Positive Pass

3.1 Hidden Layer Activation for Positive Sample

For the positive pass, we pass the real input $x = \begin{bmatrix} 0.9 \\ 0.1 \\ 0.8 \end{bmatrix}$ through the network:

$$z^{(1)} = W^{(1)}x + b^{(1)} = \begin{bmatrix} 0.2 & 0.4 & 0.6 \\ 0.8 & 0.1 & 0.3 \end{bmatrix} \begin{bmatrix} 0.9 \\ 0.1 \\ 0.8 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.07 \end{bmatrix}$$

Now apply the activation function (sigmoid):

$$a^{(1)} = f(z^{(1)}) = \begin{bmatrix} \frac{1}{1+e^{-1.2}} \\ \frac{1}{1+e^{-1.07}} \end{bmatrix} \approx \begin{bmatrix} 0.7685 \\ 0.7443 \end{bmatrix}$$

3.2 Output Layer Activation for Positive Sample

For the output layer, the activation is computed as:

$$z^{(2)} = W^{(2)}a^{(1)} + b^{(2)} = \begin{bmatrix}0.3 & 0.7\end{bmatrix} \begin{bmatrix}0.7685 \\ 0.7443\end{bmatrix} + 0.2 = 0.95156$$

Apply the sigmoid activation function:

$$\hat{y}^{\text{positive}} = f(z^{(2)}) = \frac{1}{1 + e^{-0.95156}} \approx 0.7214$$

4 Negative Pass

4.1 Hidden Layer Activation for Negative Sample

Now, let's pass the negative sample $x_{\text{neg}} = \begin{bmatrix}0.2 \\ 0.5 \\ 0.1\end{bmatrix}$ through the network:

$$z^{(1)}_{\text{neg}} = W^{(1)}x_{\text{neg}} + b^{(1)} = \begin{bmatrix}0.2 & 0.4 & 0.6 \\ 0.8 & 0.1 & 0.3\end{bmatrix} \begin{bmatrix}0.2 \\ 0.5 \\ 0.1\end{bmatrix} + \begin{bmatrix}0.5 \\ 0.1\end{bmatrix} = \begin{bmatrix}0.8 \\ 0.34\end{bmatrix}$$

Apply the activation function:

$$a^{(1)}_{\text{neg}} = f(z^{(1)}_{\text{neg}}) = \begin{bmatrix}\frac{1}{1+e^{-0.8}} \\ \frac{1}{1+e^{-0.34}}\end{bmatrix} \approx \begin{bmatrix}0.6899 \\ 0.5844\end{bmatrix}$$

4.2 Output Layer Activation for Negative Sample

For the output layer:

$$z^{(2)}_{\text{neg}} = W^{(2)}a^{(1)}_{\text{neg}} + b^{(2)} = \begin{bmatrix}0.3 & 0.7\end{bmatrix} \begin{bmatrix}0.6899 \\ 0.5844\end{bmatrix} + 0.2 = 0.81605$$

Apply the sigmoid activation:

$$\hat{y}^{\text{negative}} = f(z^{(2)}_{\text{neg}}) = \frac{1}{1 + e^{-0.81605}} \approx 0.6934$$

5 Goodness Score Calculation

For each layer, we compute a goodness score, which is the sum of squared activations.

For the hidden layer (positive sample):

$$G^{(1)}_{\text{positive}} = (0.7685)^2 + (0.7443)^2 \approx 0.5906 + 0.554 \approx 1.1446$$

For the hidden layer (negative sample):

$$G^{(1)}_{\text{negative}} = (0.6899)^2 + (0.5844)^2 \approx 0.4759 + 0.3415 \approx 0.8174$$

6 Weight Updates

6.1 Hidden Layer

To update the weights, we aim to increase the goodness score for the positive sample and decrease it for the negative sample. We can set a threshold θ and use sigmoid function σ to form a logistic regression problem:

$$p_{\text{positive}}(G) = \sigma(G - \theta)$$

The update rule for weights $W^{(1)}$ is:

$$W^{(1)} = W^{(1)} + \eta \frac{\partial p_{\text{positive}}(G^{(1)}_{\text{positive}})}{\partial W^{(1)}} - \eta \frac{\partial p_{\text{positive}}(G^{(1)}_{\text{negative}})}{\partial W^{(1)}}$$

Where η is the learning rate. The gradients are computed based on the activations and goodness scores. However, the exact weight update equations would depend on the specific implementation.

6.2 Output Layer

For output layer, we use supervised loss just the same as backpropagation. **However, since this propagation only happens in one layer, there is actually no "Back"!**

7 Conclusion

The forward-forward algorithm offers a biologically plausible way to train neural networks by comparing forward passes with real and synthetic data. In this example, we saw how concrete positive and negative samples were used to update weights, bypassing the need for back-propagation. This algorithm could simplify neural network training and provide insights into more biologically inspired models of learning.