Deep Feedforward Neural Networks

Nonlinear Representation Learning and Backpropagation

Yasser Hifny

Faculty of Computers and Artificial Intelligence
Helwan University

Contents

1. Motivation
2. Activation Functions
3. Forward Propagation
4. Backpropagation Overview
5. XOR Example
6. Assignment
7. Summary

Motivation

Deep feedforward neural networks consist of multiple layers of neurons that can learn complex patterns and representations from input data.

If every layer uses a linear activation, stacking layers does not increase the expressive power:

$$y = W^L W^{L-1} \cdots W^1 x = Wx.$$

Therefore, the complete network is still only a linear transformation of the input.

Key idea

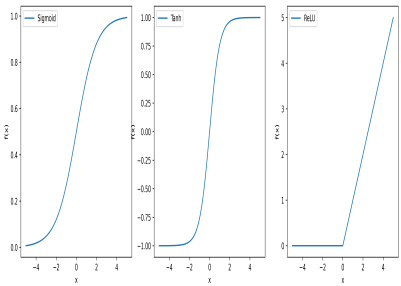
Nonlinear activation functions are required to make a deep network capable of representing nonlinear relationships.

Nonlinear Activation Functions

We introduce nonlinear functions such as:

- sigmoid,
- tanh,
- ReLU.

They can map inputs into different ranges and allow the network to form nonlinear decision boundaries.



Three activation functions commonly used in deep networks.

Deep Network with Nonlinear Activations

With nonlinear activations, a deep network can be written as

$$y = W^L g\big(W^{L-1} \cdots g\big(W^2 g(W^1 x + b^1) + b^2\big) \cdots + b^{L-1}\big) + b^L.$$

- Each layer performs a linear transformation followed by a nonlinear transformation.
- The composition of these transformations creates expressive representations.
- Nonlinear networks can approximate continuous functions.

Why XOR Requires Nonlinearity

The XOR truth table is

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0

A linear decision boundary would require

$$x_1 w_1 + x_2 w_2 \lessgtr \theta.$$

For $(1, 0)$ and $(0, 1)$ to be positive:

$$w_1 > \theta, \qquad w_2 > \theta.$$

But $(1, 1)$ must be negative:

$$w_1 + w_2 < \theta,$$

which contradicts the first two inequalities.

XOR: Consequence

There is no single linear decision boundary that separates the four XOR points into the required two classes.

Therefore, the XOR problem requires a nonlinear neural network with at least one hidden layer.

$$\boxed{\text{Linear layers alone} \not\Rightarrow \text{nonlinear decision boundaries}}$$

$$\boxed{\text{Hidden layer} + \text{nonlinear activation} \Rightarrow \text{nonlinear representation}}$$

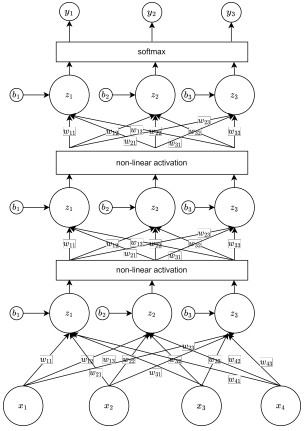
Output Layer and Learning Task

The output layer is selected according to the prediction task.

Task	Output activation	Loss function
Regression	Linear	Mean squared error
Binary classification	Sigmoid	Binary cross-entropy
Multiclass classification	Softmax	Categorical cross-entropy
Multilabel classification	Sigmoid	K Binary cross-entropy

Our main DNN example uses a softmax output for a multiclass classification problem.

Deep Feedforward Neural Network



Two hidden layers, nonlinear hidden activations, and a three-node softmax output layer.

Network Model

Consider a network with two hidden layers and an output layer.

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}, \qquad a^{[l]} = \text{sigmoid}(z^{[l]})$$

for the hidden layers, with

$$a^{[3]} = \text{softmax}(z^{[3]}).$$

- $\boldsymbol{x}$: input vector.
- $W^{[1]}, W^{[2]}, W^{[3]}$: weight matrices.
- $b^{[1]}, b^{[2]}, b^{[3]}$: bias vectors.
- $a^{[0]} = \boldsymbol{x}$.

Forward Propagation

Hidden layer 1

$$z^{[1]} = W^{[1]}\boldsymbol{x} + b^{[1]}, \qquad a^{[1]} = \text{sigmoid}(z^{[1]}).$$

Hidden layer 2

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \qquad a^{[2]} = \text{sigmoid}(z^{[2]}).$$

Output layer

$$z^{[3]} = W^{[3]}a^{[2]} + b^{[3]}, \qquad a^{[3]} = \text{softmax}(z^{[3]}).$$

Learning via Backpropagation

Backpropagation trains the network by combining a forward pass with a backward pass.

1. Propagate the input forward through all layers.
2. Compute the prediction error using the target and output.
3. Propagate the error backward using the chain rule.
4. Compute gradients for weights and biases.
5. Update the parameters using gradient descent.

$$\boxed{\text{Forward pass} \rightarrow \text{Loss} \rightarrow \text{Backward pass} \rightarrow \text{Update}}$$

Backpropagation 1: Output Layer and Loss

For K classes, the softmax output is

$$a_i^{[3]} = \frac{e^{z_i^{[3]}}}{\sum_{j=1}^{K} e^{z_j^{[3]}}}.$$

The outputs form a probability distribution:

$$a_i^{[3]} \geq 0, \qquad \sum_{i=1}^{K} a_i^{[3]} = 1.$$

For a one-hot target t, the cross-entropy loss is

$$E = -\sum_{i=1}^{K} t_i \log a_i^{[3]}.$$

Backpropagation 2: Gradient at the Output

By the chain rule:

$$\frac{\partial E}{\partial z} = \frac{\partial \mathbf{a}}{\partial \mathbf{z}} \frac{\partial E}{\partial \mathbf{a}}, \qquad \frac{\partial E}{\partial \mathbf{a}} = \begin{pmatrix} -t_1/a_1 \\ \vdots \\ -t_K/a_K \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ -1/a_j \\ \vdots \\ 0 \end{pmatrix}$$

where j is the index of the correct class. The softmax Jacobian is

$$\frac{\partial \mathbf{a}}{\partial \mathbf{z}} = \begin{pmatrix} a_1 & 0 & 0 & \cdots & 0 \\ 0 & a_2 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & a_K \end{pmatrix} - \begin{pmatrix} a_1 a_1 & a_1 a_2 & a_1 a_3 & \cdots & a_1 a_K \\ a_2 a_1 & a_2 a_2 & a_2 a_3 & \cdots & a_2 a_K \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_K a_1 & a_K a_2 & a_K a_3 & \cdots & a_K a_K \end{pmatrix} \tag{1}$$

Backpropagation 2: Gradient at the Output

$$\frac{\partial \mathbf{a}}{\partial \mathbf{z}} = \text{diag}(a_1, \ldots, a_K) - \mathbf{a}\mathbf{a}^T$$

Hence

$$\frac{\partial E}{\partial z} = \frac{1}{a_j} \left[\begin{pmatrix} a_1 a_j \\ \vdots \\ a_j a_j \\ \vdots \\ a_K a_j \end{pmatrix} - \begin{pmatrix} 0 \\ \vdots \\ a_j \\ \vdots \\ 0 \end{pmatrix} \right] = \mathbf{a} - \mathbf{t}$$

Backpropagation 3: Gradients for All Layers

Output layer:

$$\delta^{[3]} = a^{[3]} - t, \qquad \frac{\partial E}{\partial W^{[3]}} = \delta^{[3]} (a^{[2]})^T, \qquad \frac{\partial E}{\partial b^{[3]}} = \delta^{[3]}$$

Hidden layer 2:

$$\delta^{[2]} = (W^{[3]})^T \delta^{[3]} \circ a^{[2]} \circ (1 - a^{[2]}), \qquad \frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} (a^{[1]})^T, \qquad \frac{\partial E}{\partial b^{[2]}} = \delta^{[2]}$$

Hidden layer 1:

$$\delta^{[1]} = (W^{[2]})^T \delta^{[2]} \circ a^{[1]} \circ (1 - a^{[1]}), \qquad \frac{\partial E}{\partial W^{[1]}} = \delta^{[1]} x^T, \qquad \frac{\partial E}{\partial b^{[1]}} = \delta^{[1]}$$

Here, $\circ$ denotes element-wise multiplication (the Hadamard product).

Backpropagation 4: Deriving the Delta Terms

The delta term is the gradient of the loss with respect to the pre-activation of layer l:

$$\delta^{[l]} = \frac{\partial E}{\partial z^{[l]}}, \qquad \delta^{[3]} = \frac{\partial E}{\partial a^{[3]}} \cdot \frac{\partial a^{[3]}}{\partial z^{[3]}} = a^{[3]} - t$$

For hidden layer 2, the loss depends on $z^{[2]}$ indirectly through $z^{[3]}$:

$$\frac{\partial E}{\partial z^{[2]}} = \frac{\partial E}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}}$$

Since $z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}$, we have $\partial z^{[3]} / \partial a^{[2]} = W^{[3]}$. In vector form, with $a^{[2]} = g(z^{[2]})$:

$$\delta^{[2]} = (W^{[3]})^T \delta^{[3]} \circ g'(z^{[2]})$$

For the sigmoid, $g'(z) = g(z)\,(1 - g(z))$, so

$$\boxed{\delta^{[2]} = (W^{[3]})^T \delta^{[3]} \circ a^{[2]} \circ (1 - a^{[2]})}$$

Hidden layer 1 is computed the same way, using $\delta^{[2]}$ and $W^{[2]}$.

Backpropagation 5: Gradient Descent Update

The weights and biases are updated by averaging the gradients over N samples:

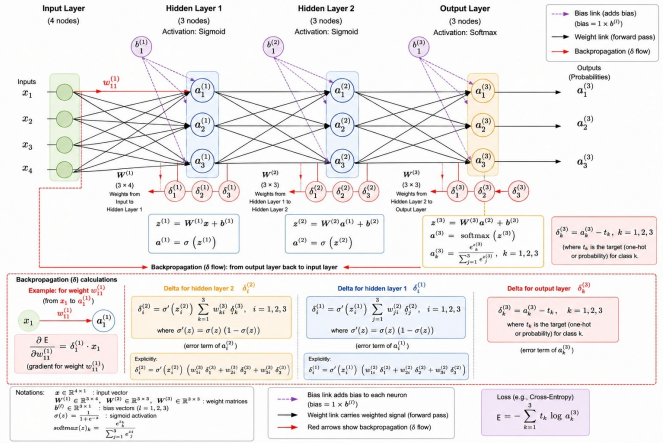
$$W^{[l]} = W^{[l]} - \eta \frac{1}{N} \sum_{n=1}^{N} \frac{\partial E^n}{\partial W^{[l]}}, \tag{2}$$

$$b^{[l]} = b^{[l]} - \eta \frac{1}{N} \sum_{n=1}^{N} \frac{\partial E^n}{\partial b^{[l]}}, \tag{3}$$

where η is the learning rate.

- These updates are performed after each iteration (or epoch, for full-batch training).
- The process is repeated for multiple epochs until the network is adequately trained.

Backpropagation Overview



Forward propagation computes activations; the backward pass propagates error signals and computes gradients.

XOR Example

We provide a NumPy implementation to learn XOR using a two-layer nonlinear network.

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

Architecture:

- 2 input variables,
- 4 hidden units,
- 1 sigmoid output,
- random initialization,
- learning rate 0.1,
- $10,000$ iterations.

XOR Example: Forward and Backward Pass

```
for epoch in range(epochs):
    # Forward propagation
    l0 = X
    l1 = sigmoid(np.dot(l0, w0) + b0)
    l2 = sigmoid(np.dot(l1, w1) + b1)

    # Backward propagation
    l2_error = Y - l2
    l2_delta = l2_error * sigmoid_derivative(l2)

    l1_error = l2_delta.dot(w1.T)
    l1_delta = l1_error * sigmoid_derivative(l1)

    # Update parameters
    w1 += l1.T.dot(l2_delta) * lr
    b1 += np.sum(l2_delta, axis=0, keepdims=True) * lr
    w0 += l0.T.dot(l1_delta) * lr
    b0 += np.sum(l1_delta, axis=0, keepdims=True) * lr
```

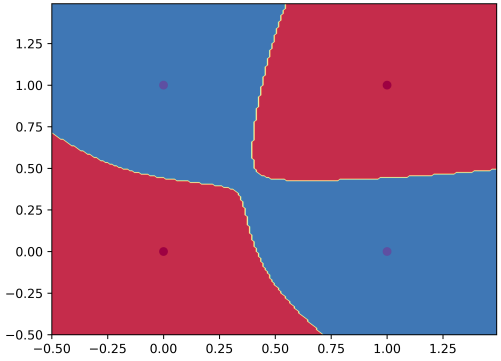
XOR Example: Decision Boundary Code

```
# Plot decision boundary
h = 0.01
x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
xx, yy = np.meshgrid(
    np.arange(x_min, x_max, h),
    np.arange(y_min, y_max, h))

l0 = np.c_[xx.ravel(), yy.ravel()]
l1 = sigmoid(np.dot(l0, w0) + b0)
l2 = sigmoid(np.dot(l1, w1) + b1)

Z = (l2 > 0.5).astype(int)
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Spectral)
plt.scatter(X[:, 0], X[:, 1], c=Y[:, 0], cmap=plt.cm.Spectral)
plt.savefig('xor_decision_boundary.png', dpi=600)
```

XOR Decision Boundary



The learned nonlinear boundary separates $(0, 0)$ and $(1, 1)$ from $(0, 1)$ and $(1, 0)$.

From Linear to Deep Networks

Linear layers

Composition remains linear.

Nonlinear activations

Intermediate representations become nonlinear.

Backpropagation

Gradients train the parameters of all layers.

$$\boxed{\text{Deep network} = \text{composition of transformations} + \text{nonlinearities}}$$

Assignment

Implement optical recognition of handwritten digits as a multiclass classification task using a deep feedforward neural network (using Python from scratch, using Keras tookit, and Google Colab).

Dataset:
https://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

Expected learning outcome

Apply the DNN model, nonlinear hidden layers, softmax output, categorical cross-entropy, and backpropagation to a real multiclass recognition problem.

Summary

- Stacking linear layers still produces a linear transformation.
- Nonlinear activations enable expressive representations and nonlinear decision boundaries.
- XOR demonstrates why a hidden nonlinear layer is necessary for some problems.
- A DNN can use different output activations and losses according to the task.
- Forward propagation computes the network output.
- Backpropagation applies the chain rule to compute layer-wise gradients.
- The delta terms propagate the error backward through the network.
- Gradient descent updates weights and biases over repeated epochs.

Thank You

Questions?