Module 5: The Deep Learning & Hugging Face Ecosystem, Lesson 1: What is Deep Learning?

Hands-On: Building a Neuron from Scratch — Live NumPy Demo

Overview

This demo proves the three core claims from the lecture in real time, using nothing but NumPy and about eighty lines of Python:

- A single neuron is just a dot product plus a non-linearity — there is no magic, only math you already know.
- A single-layer network cannot learn XOR — this is the historical wall that paused AI research for a decade.
- Adding one hidden layer breaks the wall — depth is what unlocks expressive power, and that is the entire premise of deep learning.

You will run three small Python programs in sequence. The student assignment at the end of this hands-on extends the third program by changing weights and activations to build intuition for how the pieces interact.

Before You Start

- Have Python 3.10+ installed. Verify with python3 --version.
- Install NumPy: pip install numpy.
- Open the file lecture1_neuron_demo.py from https://github.com/AI-Native-Engineer-Course/AI-Native-Engineer/tree/main/module 5/lecture 1.
- Have a terminal open in the same directory, ready to run the file step by step.
- Have the lecture slides visible on a second screen if possible — the visual of the XOR truth table makes the second step land harder.

> **REMEMBER**
> Every deep learning concept in this course rests on these eighty lines. A neuron is a weighted sum followed by a non-linearity. Stack them and you get a network. Train them and you get a model. Nothing more, nothing less. Watch the pattern repeat in every lecture that follows.

During Hands-On — Three Steps at a Glance

- STEP 1 — One neuron, one forward pass. The dot product made concrete. ~4 min
- STEP 2 — The XOR wall. Watch a single layer fail to learn a problem a child solves in seconds. ~4 min
- STEP 3 — Add a hidden layer. Same data, same activation — and now it works. ~5 min

Step 1 — The Forward Pass of a Single Neuron

WHAT TO LEARN

A neuron takes a vector of inputs, multiplies each by a weight, sums them, adds a bias, and pushes the result through a non-linear function. That is the entire computation. The non-linearity is what lets us eventually compose neurons into networks that learn — without it, no matter how many layers you stack, you collapse back into a single linear function.

Open lecture1_neuron_demo.py and find the section labeled PART 1. Run it from the terminal:

```
python3 lecture1_neuron_demo.py
```

Read the code with the room. Three lines do all the work:

```
inputs   = np.array([0.5, -1.2, 3.1])
weights  = np.array([0.4,  0.8, -0.5])
bias     = 0.1
z        = np.dot(inputs, weights) + bias
output   = sigmoid(z)
```

Flowchart of part 1.

WHAT YOU SHOULD GET

Two printed lines: the pre-activation z (around -2.65) and the post-activation output (a sigmoid-squashed value near 0.066). Point at the numbers. Tell the room: this is exactly what one neuron does inside every transformer, every CNN, every model on Hugging Face. The scale changes — billions of these instead of one — but the operation is identical.

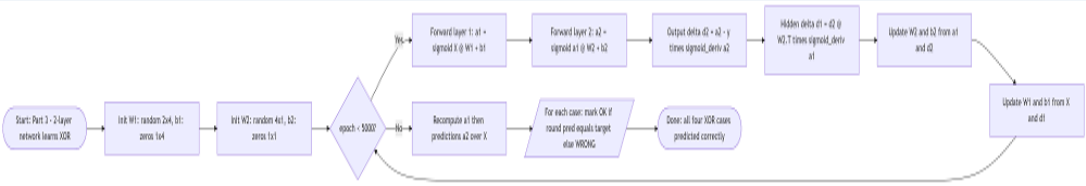
Step 2 — Why a Single Neuron Fails on XOR

WHAT TO LEARN

> XOR returns 1 when exactly one input is 1 — the classic non-linearly-separable problem. No single straight line through the (x1, x2) plane can separate the two output classes. A single neuron can only draw a straight line. So no single neuron, no matter how you tune its weights, can solve XOR. This is the result that froze AI research in the 1970s.

Find PART 2 in the same file. It loops over the four XOR inputs and tries to find any single-neuron weights that classify all four correctly:

```
XOR_DATA = [
    ([0, 0], 0),
    ([0, 1], 1),
    ([1, 0], 1),
    ([1, 1], 0),
]
```



Flowchart for part 3.

Run the file again and read the output as a class:

```
python3 lecture1_neuron_demo.py
```

> **WHAT YOU SHOULD GET**
> The output for the four inputs will not match the truth table no matter what weights the search picks. The best single-neuron classifier gets three out of four. That fourth case is the wall. Pause here for the room. This is the moment in the lecture that motivates everything else in the module.

You should ask the room:

"Why does this matter today?"

Answer: because every classification problem worth solving is non-linearly-separable. Recognizing a cat in a photo is XOR with millions of dimensions. If a single layer cannot do XOR, it cannot do vision, or language, or weather forecasting either.

Step 3 — A Two-Layer Network Solves It

WHAT TO LEARN

Add one hidden layer with two neurons and a non-linear activation between them, and the problem dissolves. The hidden layer learns to project the input into a new space where XOR becomes linearly separable. The output neuron then draws the dividing line in that new space. This is the entire intuition behind why depth works — each layer learns a richer representation of the input.

Find PART 3 in the file. The network is hand-built — no training loop, no gradient descent yet. Weights and biases are set by hand to known values that solve XOR:

```
# Hidden layer: 2 inputs -> 2 neurons
W1 = np.array([[ 1.0,  1.0],
               [ 1.0,  1.0]])
b1 = np.array([-0.5, -1.5])

# Output layer: 2 hidden -> 1 output
W2 = np.array([ 1.0, -1.0])
b2 = -0.5
```

Run the file one more time. The output now matches the XOR truth table exactly across all four inputs.

```
python3 lecture1_neuron_demo.py
```

WHAT YOU SHOULD GET

Four lines of output. Inputs (0,0) and (1,1) produce values near 0. Inputs (0,1) and (1,0) produce values near 1. That is XOR — solved by hand-tuned weights in a two-layer network.

Wrap-Up — The Pattern, The Assignment Handoff

REMEMBER

A neuron is a weighted sum plus a non-linearity. A single layer cannot solve XOR — or anything else worth solving. Adding depth changes everything because each layer reshapes the input space until the problem becomes simple enough for a straight line to separate. Every model in this course — including the billion-parameter ones on Hugging Face — is built on this exact pattern, repeated at scale.

Key takeaways:

- The forward pass of a neuron is dot(inputs, weights) + bias, run through a non-linearity. That is the whole computation.
- Single-layer networks are limited to linearly-separable problems — a small subset of useful tasks.
- Depth gives a network the ability to learn its own feature representation, which is why deep learning replaced classical machine learning for vision and language.
- The non-linearity between layers is non-negotiable — drop it and the network collapses back to a single linear function regardless of depth.

Source File Reference

The full source for this demo lives at https://github.com/AI-Native-Engineer-Course/AI-Native-Engineer/tree/main/module 5/lecture 1/lecture1_neuron_demo.py in the course repository. It is roughly eighty lines, all standard NumPy, no external dependencies beyond numpy itself. Open it once before class and walk through the three PART markers so you know where to pause for questions.