ExcelML Example01: Learning XOR End-to-End in Excel (Step-by-Step)

This walkthrough is written for **Excel-native users**. You will be told **exactly what values and formulas to enter into which cells**, and why each step matters.

You do *not* need prior machine learning knowledge.

By the end, you will:

- Train a real neural network inside Excel
- See learning happen numerically and visually
- Understand *why* XOR requires a hidden layer
- Inspect the internal state of the model

Conceptual anchor (keep this in mind)

In ML terms, you are:

> Training a neural network using gradient descent

In Excel terms, you are:

> Using cells and formulas to explicitly control a learning process

Every formula you enter corresponds to a real ML operation.

Step 0 — Create a new worksheet

1. Open a **new blank workbook** in Excel
2. Use the **first worksheet** (Sheet1)
3. Ensure the ExcelML add-in is loaded

Step 1 — Enter the XOR dataset

We start by entering the smallest dataset that *cannot* be solved by a straight line.

1.1 Enter headers

Cell	Value
A1	X1
B1	X2
C1	y

1.2 Enter data

Cell	Value
A2	0
B2	0
C2	0
A3	0
B3	1
C3	1
A4	1
B4	0
C4	1
A5	1
B5	1
C5	0

At this point:

- **A2:B5** is your input matrix X
- **C2:C5** is your output vector y

Step 2 — Create a model

We now tell ExcelML what kind of neural network we want.

2.1 Choose a cell for the model ID

We will store the model ID in **E2**.

2.2 Enter the formula

In **cell E2**, type:

```
=DL.MODEL_CREATE("xor:in=2,hidden=8,out=1")
```

2.3 What you should see

- The cell will display a long string (a model ID)
- This string uniquely identifies a model stored in memory

2.4 Conceptual meaning

- `in=2` → two input features (X1, X2)
- `hidden=8` → one hidden layer with 8 neurons
- `out=1` → one output value (probability of class 1)

Excel is now holding a *reference* to a real neural network.

Step 3 — Create a training trigger

Excel recalculates formulas automatically. Training a neural network should **not** happen automatically.

We solve this with a trigger cell.

3.1 Choose a trigger cell

We will use **Z1**.

3.2 Enter an initial value

In **cell Z1**, type:

```
1
```

This value has no meaning except:

> "Train when this value changes."

Step 4 — Train the model

This is the most important step.

4.1 Choose a cell for training output

We will use **E4**.

4.2 Enter the training formula

In **cell E4**, type:

```
=DL.TRAIN(E2, A2:B5, C2:C5, "epochs=1000,lr=0.1", $Z$1)
```

4.3 What this formula means

Argument	Meaning
E2	Model ID
A2:B5	Input data X
C2:C5	Target values y
epochs=1000	Number of training passes
lr=0.1	Learning rate
Z1	Training trigger

4.4 What you should see

- Training runs once
- The cell displays a small table with status, epochs, and final loss

If you **recalculate Excel**, training does *not* run again.

Step 5 — View the loss history

Loss tells us how wrong the model is.

5.1 Choose a cell

We will start the loss table in **E8**.

5.2 Enter the formula

In **cell E8**, type:

```
=DL.LOSS_HISTORY(E2)
```

5.3 What you should see

A spilled table like:

epoch	loss
1	~0.69
...	...
1000	much smaller

5.4 Optional: create a chart

1. Select the loss table
2. Insert → Line Chart

You are now watching a neural network learn.

Step 6 — Retrain on demand

6.1 Change the trigger

Change **Z1** from `1` to `2`.

6.2 What happens

- Training runs again
- Loss history resets
- New weights are learned

This mirrors real ML experimentation.

Step 7 — Inspect the model (advanced but illuminating)

You can look *inside* the model.

7.1 View weights

In an empty area, type:

```
=DL.WEIGHTS(E2, "fc1")
```

7.2 View gradients

```
=DL.GRADS(E2, "fc1")
```

7.3 View activations

```
=DL.ACTIVATIONS(E2, A2:B5, "fc1")
```

Each of these spills a table you can analyze or chart.

What you just did

Without writing code, you:

- Built a neural network
- Trained it using gradient descent

- Controlled training explicitly
- Visualized learning
- Inspected internal representations

Why XOR matters

XOR proves:

- Linear models are limited
- Hidden layers create expressive power

ExcelML lets you *see* this — not just read about it.

Next experiments

Try:

- Changing `hidden=2`, `hidden=16`
- Lowering or raising `lr`
- Removing the hidden layer

Each change teaches a real ML lesson.

Appendix: Conceptual and Pedagogical Deep Dive

This appendix expands on the walkthrough by explaining **what is really happening conceptually** at each stage, and *why the design choices in ExcelML matter pedagogically*.

It is intended for readers who want to move beyond “how” and into “why”.

A1. Why XOR is the perfect first neural network problem

XOR is not chosen because it is small — it is chosen because it is *revealing*.

Linear intuition breaks

A linear model tries to separate classes with a straight line (or plane). XOR cannot be separated that way:

- (0,0) and (1,1) belong to one class
- (0,1) and (1,0) belong to the other

No straight line can separate these two sets.

What XOR teaches

XOR demonstrates, in the smallest possible form:

- the limitation of linear models
- the necessity of hidden layers
- the idea of representation learning

When learners see XOR solved *only after adding a hidden layer*, the motivation for neural networks becomes concrete.

A2. What a neural network layer really is

A neural network layer performs three steps:

1. **Weighted sum** of inputs
2. **Add a bias**
3. **Apply a non-linear function**

In Excel terms:

- weights behave like coefficients
- bias behaves like an intercept
- the activation function bends space

Hidden layers are not magic — they are *coordinate transformations*.

A3. Why non-linearity matters

If you stack linear layers without non-linear activations, the result is still linear.

This is why:

- removing `Tanh` (or ReLU) causes XOR training to fail
- adding more linear layers does nothing

Non-linearity allows the model to:

- bend decision boundaries
- carve complex regions
- represent conditional logic

ExcelML makes this visible when you inspect activations.

A4. Gradient descent, intuitively

Gradient descent is often explained with calculus, but conceptually it is simple:

> *Take a small step in the direction that reduces error the most.*

In ExcelML:

- each epoch is one full pass over the data
- the loss measures “how wrong” the model is
- gradients tell each weight which direction to move

By inspecting `DL.GRADS`, you can literally see:

- which weights are changing
- which ones matter

A5. Why loss curves matter more than final accuracy

Beginners often focus on final results. Experts watch **learning curves**.

Loss curves tell you:

- whether learning is happening
- whether learning is stable

- whether the learning rate is too high or too low

In Excel, charts make these patterns visually obvious.

This is why ExcelML emphasizes `DL.LOSS_HISTORY` early.

A6. The role of the learning rate

The learning rate controls step size:

- Too small → learning is slow
- Too large → learning becomes unstable

Encouraged experiment:

- Try `lr=0.01`, `lr=0.1`, `lr=1.0`
- Observe how the loss curve changes

ExcelML turns hyperparameter tuning into a visual exercise.

A7. Why explicit training triggers matter pedagogically

Most ML tools hide training behind execution flow.

ExcelML makes training **explicit and intentional**:

```
=DL.TRAIN(..., trigger)
```

This teaches:

- training is a *decision*, not a side-effect
- models have state
- recomputation ≠ retraining

This mental model transfers directly to production ML systems.

A8. Inspectability vs black boxes

Traditional tools optimize for convenience.

ExcelML optimizes for:

- visibility
- traceability
- causal understanding

Being able to inspect:

- weights
- gradients
- activations

changes learning from *belief* to *observation*.

A9. Why Excel is uniquely suited for this

Excel offers:

- deterministic recalculation
- immediate feedback
- structured grids
- built-in visualization

These align unusually well with ML pedagogy.

ExcelML does not fight Excel — it leans into it.

A10. Transfer of learning

Nothing in ExcelML is toy-only.

The concepts learned here map directly to:

- PyTorch
- TensorFlow
- real-world ML pipelines

ExcelML is not a replacement — it is a **ramp**.

Closing thought

Most ML confusion comes from abstraction layers that hide causality.

ExcelML removes those layers.

> **Understanding comes from seeing cause and effect.**

This appendix exists to help you build that intuition — deliberately, visually, and one cell at a time.