

Inside a Neural Network

Six components, one learning loop, and what 23 training runs in TensorFlow Playground taught me

Bindu Bhargava Reddy Chintam | AIML-501: Model Development | Indiana Wesleyan University | October 2026

Six parts, three jobs

I sorted the six components by the job each one does. That grouping is the backbone of this deck.

1 Structure

Where data goes

Layers

Stages that transform the data

Neurons

Units that each do one small calculation

2 Signal

How data is shaped

Weights and biases

Dials that set each input's influence

Activation functions

Bends that let the network learn curves

3 Learning loop

How the network improves

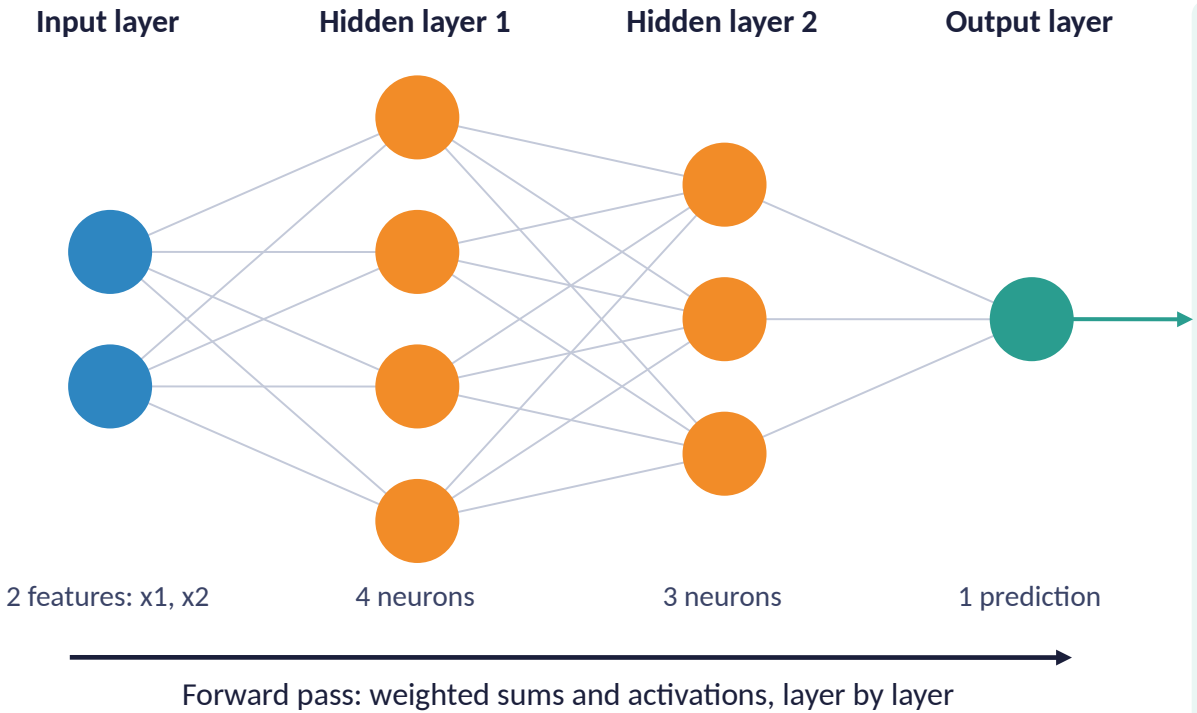
Loss function

Scores how far off a prediction is

Optimizer

Rule that nudges weights to lower the loss

How data flows through the network



Learning loop

1. Loss: compare prediction with the true label
2. Gradients: trace the error back through each weight
3. Optimizer: nudge every weight a small step
4. Repeat for the next batch

Layers: stages that build features

I

Input layer

Receives the raw features, such as a point's x and y position. It does no math of its own.

H

Hidden layer

Each layer builds new features from the one before. Early layers find simple boundaries; deeper layers combine them into shapes.

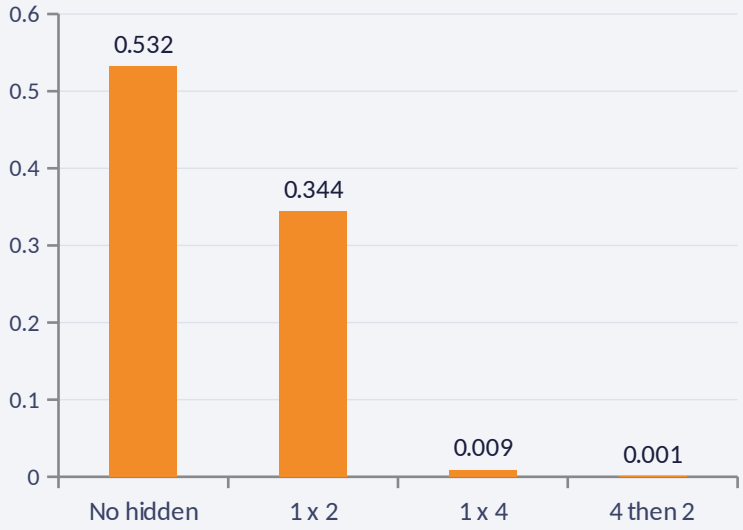
O

Output layer

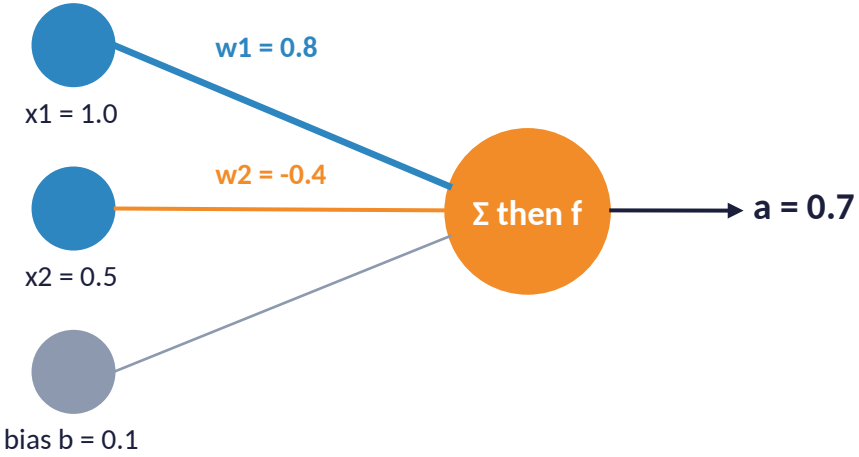
Turns the last features into the answer, such as orange or blue for a classifier.

What changed when I added layers

Circle dataset, 600 epochs, same random seed



Neurons: one small calculation each



The math

Weighted sum: $z = w1 \cdot x1 + w2 \cdot x2 + b$
$z = 0.8(1.0) + (-0.4)(0.5) + 0.1 = 0.7$

Activation: $a = f(z)$
$\text{ReLU}(0.7) = 0.7$, $\text{sigmoid}(0.7) = 0.668$

Every neuron in a layer runs this same two-step recipe with its own weights, and passes a to every neuron in the next layer.

Weights and biases: the dials that learn

w

Every connection has a weight

It scales how much one neuron's output matters to the next neuron.

w

Sign and size both matter

In Playground, blue lines are positive, orange are negative, and thicker lines are larger weights.

w

Biases shift the threshold

A bias lets a neuron turn on even when its inputs are small.

w

Training only changes these numbers

The layers and activations stay fixed; learning means finding better weights and biases.

Counting the dials

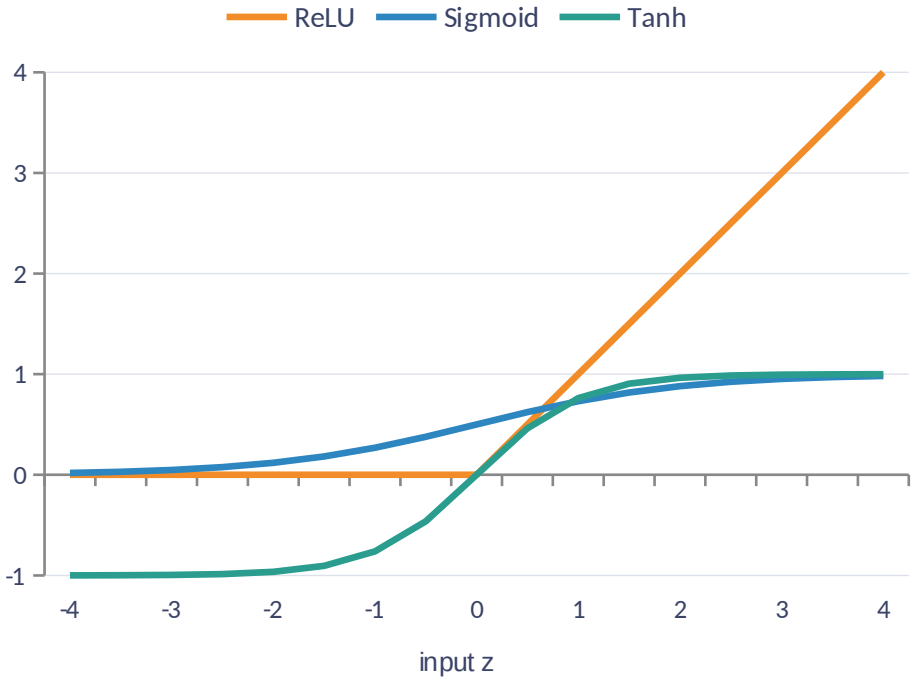
My Playground network: 2 → 4 → 2 → 1

25

learnable numbers:
(2×4 + 4) + (4×2 + 2) + (2×1 + 1)
= 12 + 10 + 3

Large language models use the same idea with billions of these numbers.

Activation functions: adding the bends



XOR data, same network, 600 epochs

Linear (no bend) — test loss 0.496
fails

Sigmoid — test loss 0.044
under 0.1 at epoch 483

Tanh — test loss 0.016
under 0.1 at epoch 60

ReLU — test loss 0.002
under 0.1 at epoch 15

Without a bend, stacking layers still gives a straight line.

Loss functions: scoring the mistakes

L

Mean squared error

For numbers (regression)

L = average of (prediction − truth)²

Big misses cost much more than small ones: a miss of 0.9 costs 0.81, a miss of 0.1 costs 0.01.

L

Cross-entropy

For classes (classification)

L = −log(probability given to the true class)

Confident mistakes are punished hard: 0.9 on the right class costs 0.105, but 0.1 costs 2.303.

In Playground: labels are -1 and +1, and the loss shown is the average of ½(output − label)². A network that guesses one class everywhere scores about 1.0. That is exactly what one of my runs did (test loss 1.032).

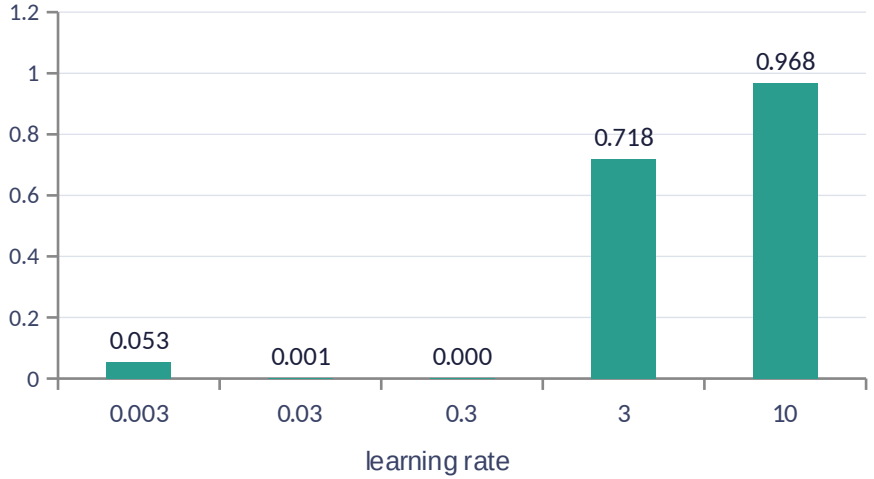
Optimizers: small steps downhill

The update rule

$$w \leftarrow w - \eta \cdot \partial L / \partial w$$

- ∂L/∂w is the slope of the loss for that weight, found by backpropagation.
- η is the learning rate: the size of each step.
- Stochastic gradient descent uses small batches; Adam also adapts the step for each weight.

Learning rate test: circle data, 600 epochs



Too small crawls (epoch 568 to get under 0.1). Too large overshoots and never settles. 0.03 to 0.3 got there by epoch 62.

One training step, all six parts together

1 — Feed in

Input layer takes a batch of points

2 — Combine

Neurons form weighted sums with their weights and biases

3 — Bend

Activation functions shape each sum

4 — Score

Loss function compares output with labels

5 — Trace back

Backpropagation finds each weight's share of the error

6 — Adjust

Optimizer updates every weight by a small step

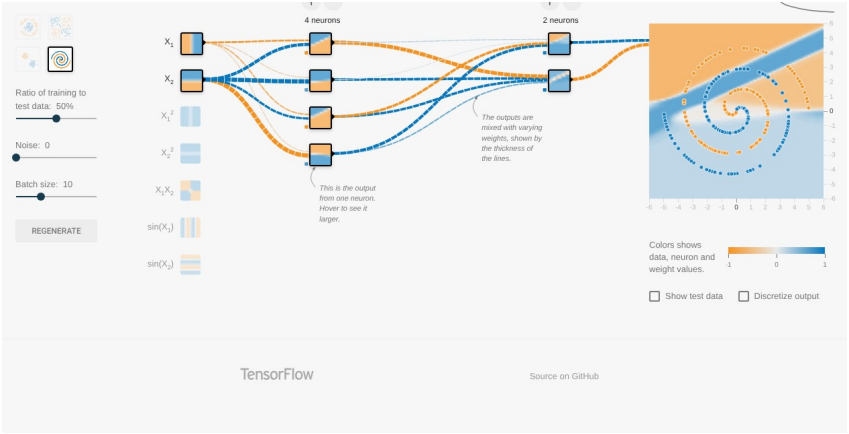
Repeat for every batch. One full pass over the training data is one epoch; my Playground runs used 600 to 1,500 epochs.

What 23 Playground runs showed

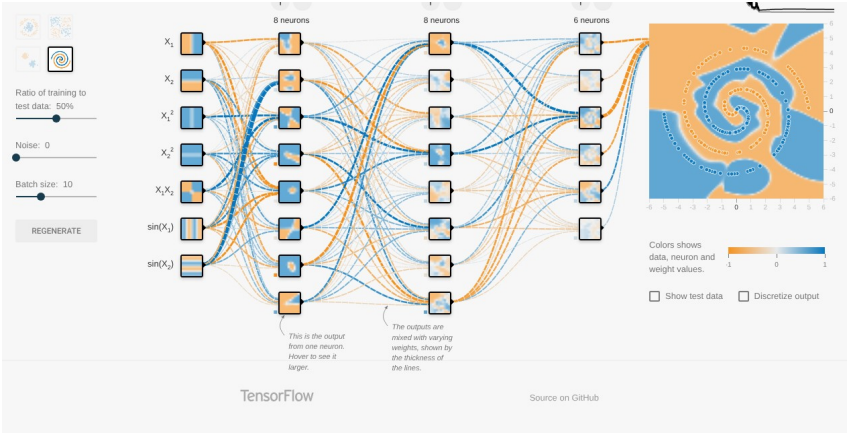
Change tested	Setting	Test loss	What it shows
Hidden layers	none vs 4 then 2	0.532	Depth lets the boundary curve (0.532 to 0.001)
Better features	$x1^2$, $x2^2$, no hidden layer	0.001	Good inputs can replace hidden layers
Random start	same setup, first seed	1.032	A bad start can freeze learning
Hard data	spiral, 4 then 2	0.468	Too small a network for the shape
Hard data	spiral, 8-8-6 + 7 features	0.045	Capacity plus features solve it (0.138 without features)
Noise	xor, noise 0 vs 50	0.275	Noise raises the floor (from 0.016)
Activation	xor, linear	0.496	No bend, no curve
Learning rate	circle, rate 3	0.718	Steps too big to settle

All runs: same seed unless noted, batch size 10, 50% of points held out for testing. Loss above 0.1 in red.

Seeing it: the spiral before and after



Small network, 2 inputs: test loss 0.468



Deeper network, 7 inputs: test loss 0.045

Screenshots from my TensorFlow Playground runs (1,500 epochs). Left: the network cannot follow the arms of the spiral. Right: extra layers and features such as sin(x1) let it trace both arms.

Why visualizing networks matters

Pictures make the math checkable

Seeing weights as colored lines and outputs as maps showed me when a network was stuck, long before the loss number said why.

Structure is a choice, not a default

The circle needed a hidden layer, unless I gave it better features. Model design starts with the data.

The learning loop can fail quietly

A bad learning rate or an unlucky start gave confident, wrong answers. Watching training is part of building responsibly.

Simple parts, big systems

The 25 dials in my small network and the billions in a language model follow the same six-part recipe.

Layers and neurons give the shape. Weights and activations carry the signal. Loss and the optimizer do the learning.

References

3Blue1Brown. (2017, October 5). But what is a neural network? | Deep learning chapter 1 [Video]. YouTube. https://www.youtube.com/watch?v=aircAruvnKk

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT Press. https://www.deeplearningbook.org

Google. (2025). Neural networks. Machine Learning Crash Course. https://developers.google.com/machine-learning/crash-course/neural-networks

IBM Technology. (2022, May 24). Neural networks explained in 5 minutes [Video]. YouTube. https://www.youtube.com/watch?v=jmmW0F0biz0

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv. https://doi.org/10.48550/arXiv.1412.6980

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323(6088), 533–536. https://doi.org/10.1038/323533a0

Smilkov, D., & Carter, S. (n.d.). TensorFlow Playground [Web application]. Retrieved October 27, 2026, from https://playground.tensorflow.org