How a Neural Network Learns

A network learns by guessing, measuring its error, and nudging its weights a little

Introduction

A neural network starts out knowing nothing. Its weights — the numbers that decide how strongly each connection counts — begin as small random values. So its first predictions are basically guesses. Learning is the process of turning those random guesses into good ones, by repeating one loop thousands of times: guess, measure how wrong the guess was, and nudge the weights a little in the direction that would have made the guess better.

Three ideas carry the whole lesson. **Weights** are the network's memory: everything it learns is stored as weights. **Loss** is a single number that says how wrong a prediction was — big loss means a bad guess, small loss means a good one. **The training loop** is the repeat-until-better cycle: forward pass (guess), loss (measure), backward pass (work out which weights to blame), update (nudge them). The backward pass has a famous name, **backpropagation**, but the idea is simple: share the blame for the error across the weights that caused it.

Nobody updates weights by hand. A method called an **optimizer** does the nudging, and the size of each nudge is set by the **learning rate**: too big and the network jumps past the good answer, too small and it crawls forever. You will see both failures in the examples below.

In this lesson you will walk the training loop by hand on a tiny example, and learn why loss, learning rate and epochs matter.

Where beginners get stuck: they wonder why weights start random instead of zero. If every weight started at zero, every unit would learn exactly the same thing — the network could never split the work between units. Random starting weights break that tie, so each unit drifts toward a different job. A second stuck point: the network never "sees" the right answer as a rule. It only sees its own error number, and nudges itself to make that number smaller. *(Roman Urdu: Network pehle andazay lagata hai, phir ghalti napta hai, aur weights ko thora thora theek karta hai — yehi loop bar bar chalne se seekhna hota hai.)*

Key points

- **Weights are what the network learns.** Input numbers are multiplied by weights, added up, and passed on. Change the weights and you change what the network predicts. Training = finding good weights.
- **A forward pass is one guess.** Data flows input → hidden layers → output. The output is the network's prediction for that example.
- **Loss measures wrongness in one number.** For a number prediction, a common loss is the squared difference between guess and truth. For categories, a common loss scores how confidently the network picked the wrong class. Lower loss = better.

- **Backpropagation shares the blame.** Starting from the loss, the network works backwards, layer by layer, computing how much each weight contributed to the error. Weights that caused more error get bigger corrections.
- **The optimizer nudges the weights.** The simplest optimizer, gradient descent, moves each weight a small step downhill on the loss — like walking down a hill in fog, feeling the slope under your feet.
- **Learning rate = step size.** Too large: the network overshoots and the loss jumps around or explodes. Too small: training crawls and may never finish. A middle value, often found by trying a few, works best.
- **An epoch = one full pass over the training data.** Networks usually train for many epochs: the same examples are shown again and again, with weights improving a little each time. Too many epochs on small data → memorisation (overfitting).
- **Training usually uses small batches, not one example at a time.** A batch (for example 32 examples) gives a steadier error signal than a single example, and it uses the computer's parallel power well. The weights are nudged once per batch, not once per example.
- **Read the loss curve, not just the final number.** A smooth falling curve means healthy learning. A flat curve means the network stopped learning; a jumping curve usually means the learning rate is too big; a curve that falls on training data but rises on validation data means overfitting.

Examples

Example 1 — Learning one weight by hand

A tiny network predicts a score from hours studied: prediction = weight × hours. Truth: 2 hours should give a score of 10, so the true weight is 5. The network starts at weight = 1.

- Forward pass: $1 \times 2 = 2$. Loss: $(10 - 2)^2 = 64$. Bad guess.
- Backward pass: the weight is too small — increasing it would reduce the loss.
- Update (learning rate 0.5, simplified): weight becomes $1 + 0.5 \times 8 = 5$. Next guess: $5 \times 2 = 10$. Loss: 0. Learned in one step, because the problem is tiny.

Real networks have millions of weights and messy data, so they need thousands of such steps, but the loop is exactly the same: guess, measure, blame, nudge.

Example 2 — Learning rate too big, too small, just right

Imagine loss as a valley and the weight as a ball rolling down it.

- **Rate too big (ball leaps):** the ball jumps from one side of the valley to the other, higher each time. Loss goes up, not down. Training fails loudly.
- **Rate too small (ball crawls):** the ball moves a hair each step. After your whole training time it is still far from the bottom. Training fails quietly.
- **Rate just right:** the ball rolls steadily downhill and settles near the bottom. Loss falls smoothly, epoch after epoch.

Practical habit: try a few learning rates (for example 0.1, 0.01, 0.001), watch the loss curve, and keep the one that falls steadily without jumping.

Example 3 — The training loop in code (concept sketch)

```
for epoch in range(20):          # one epoch = one pass over all examples
    for x, y_true in training_data:
        y_guess = network.forward(x)      # forward pass: guess
        loss = (y_guess - y_true) ** 2    # loss: how wrong?
        network.backward(loss)            # backpropagation: share the blame
        optimizer.step(learning_rate)     # update: nudge the weights
    print("epoch", epoch, "loss", average_loss)
```

Read it top to bottom and you have the whole training loop. Real frameworks hide the middle steps, but they all run this loop.

Example 4 — Loss falling step by step (worked numbers)

Same tiny problem as Example 1: prediction = weight × hours, hours = 2, truth = 10. This time the update rule is: new weight = old weight + 0.1 × error × hours, where error = truth − guess.

- Step 1: weight 1.0, guess 2.0, error 8.0, loss $(8.0)^2 = 64$. New weight: 1.0 + 0.1 × 8.0 × 2 = 2.6.
- Step 2: weight 2.6, guess 5.2, error 4.8, loss 23.04. New weight: 2.6 + 0.1 × 4.8 × 2 = 3.56.
- Step 3: weight 3.56, guess 7.12, error 2.88, loss 8.29. New weight: 3.56 + 0.1 × 2.88 × 2 = 4.14.
- Step 4: weight 4.14, guess 8.28, error 1.72, loss 2.96.

Loss: 64 → 23 → 8 → 3. Each step is smaller than the last, because the error shrinks as the guess improves. Real training looks like this, thousands of times over, across millions of weights.

Example 5 — Early stopping: reading two curves

You train for 10 epochs and record the loss:

| Epoch | Training loss | Validation loss | |------:|--------------:|----------------:| | 1 | 1.20 | 1.25 | | 3 | 0.60 | 0.62 | | 5 | 0.35 | 0.40 | | 7 | 0.20 | 0.48 | | 10 | 0.08 | 0.70 |

Training loss keeps falling, but validation loss bottoms out near epoch 5 and then rises. The network is now memorising training examples instead of learning general patterns. The right move is early stopping: keep the weights from around epoch 5, the point where validation loss was lowest, and stop there.

Example 6 — Real world: how a voice assistant learns its wake word

A voice assistant must wake only when it hears its wake word, and ignore everything else. Its makers collect thousands of clips — the wake word spoken by many voices, plus ordinary speech that must not trigger it. Training runs the loop from this lesson over and over: guess "wake word or not", measure the loss, share the blame, nudge the weights, repeat for many epochs. The loss curve tells the team when to stop. The result is a tiny network that runs on the device all day, listening, and only wakes on the right word.

Quick recap

- Weights store everything the network learns.

- Forward pass = guess. Loss = how wrong the guess was, in one number.
- Backpropagation works out which weights deserve the blame.
- The optimizer nudges weights downhill on the loss; learning rate sets the step size.
- An epoch is one full pass over the training data; training runs for many epochs.

Common mistakes

1. **Watching accuracy instead of loss during training** — accuracy jumps in steps and hides problems; loss moves smoothly and shows learning. Fix: plot the loss curve every run.
2. **Setting the learning rate once and never questioning it** — a jumping loss usually means the rate is too big. Fix: if loss explodes, lower the rate tenfold and retry.
3. **Training for far too many epochs on small data** — the loss keeps falling on training data while the test score gets worse. Fix: keep a validation set aside and stop when its loss stops improving (early stopping).

1. **Feeding inputs on wildly different scales** — one input in the range 0–1000 and another in 0–1 makes the loss landscape lopsided and training unstable. Fix: scale inputs to a similar range (for example 0–1) before training.
2. **Showing examples in the same order every epoch** — the network can start learning the order instead of the patterns. Fix: shuffle the training data at the start of every epoch; every framework does this by default, so check it is switched on.

Practice

1. A network predicts 8, the truth is 10. What is the squared-error loss? If the next guess is 9.5, what is the new loss? Which guess is better?
2. Your loss curve jumps up and down wildly instead of falling. Name the most likely cause and the one change you would try first.
3. Explain the training loop to a friend in four short sentences: guess, measure, blame, nudge.

1. Use Example 4's rule (new weight = old weight + 0.1 × error × hours) for one more step from weight 4.14. What is the new weight, the new guess, and the new loss? Is the loss still falling?
2. Your training loss falls from 1.0 to 0.2, but your validation loss rises from 0.5 to 0.9. Name the problem and write the two changes you would try first.
3. A friend says "my network learned nothing — the loss never moved from 2.0". List two possible causes (think: learning rate, and the scale of the inputs) and the check for each.

Practice quiz

1. What does a network store everything it learns as?

- Rules written by hand
- Weights — numbers that decide how strongly each connection counts

- Screenshots
- The training data itself

Answer: Weights — numbers that decide how strongly each connection counts

2. What is loss?

- The number of layers
- One number that says how wrong a prediction was
- The training time
- The number of mistakes in the code

Answer: One number that says how wrong a prediction was

3. What happens when the learning rate is too big?

- Training gets slower
- The loss jumps around or explodes instead of falling
- Nothing changes
- The network forgets everything at once

Answer: The loss jumps around or explodes instead of falling

Watch the video lesson: https://youtu.be/-36z1B33RHU

Read online: https://scopetreasure.com/notes/dl-how-network-learns · More free tools and notes at https://scopetreasure.com