Activation Functions and Building Blocks

Without a non-linear twist, stacked layers could only draw straight lines. What

Introduction

So far a unit multiplies inputs by weights and adds them up. That is a straight-line (linear) calculation. Here is the catch: stack ten linear layers on top of each other and the whole stack is still just one straight-line calculation in disguise — like folding a long addition into a single addition. A network made only of additions and multiplications could never learn curves, corners, or "either this or that" patterns. It could only draw straight lines.

The fix is small and brilliant: after the adding-up, each unit passes its result through an **activation function** — a little bend. The bend is non-linear, which means it breaks the straight line. Once every layer bends its output, the whole network can bend, curve and fold its way around complicated patterns. Depth plus bending is what makes deep networks powerful.

This lesson introduces the famous activations, what each one is good at, and two classic failures: the vanishing gradient (signals fading to nothing in deep stacks) and dead units (units that stop learning entirely).

Where beginners get stuck: they accept that a bend is needed, but not why a straight-line stack collapses. Think of it this way: multiplying by 2, then by 3, then adding 4 is the same as multiplying by 6 and adding 4 — one straight-line step in disguise. No matter how many linear layers you stack, the whole stack can be folded into a single straight-line calculation. The bend after each layer is what stops that folding, because a bent function cannot be folded into a straight line. *(Roman Urdu: Har layer ke baad aik chhota sa mor (activation) lagta hai — isi mor ki wajah se network seedhi lakeer se zyada mushkil pattern seekh sakta hai.)*

Key points

- **An activation function bends a unit's output.** Without it, any stack of layers collapses into one straight-line calculation. With it, the network can learn curves and complex shapes.
- **Sigmoid squashes any number into 0 to 1.** It was the classic choice for "probability-like" outputs. Problem: for very large or very small inputs it goes flat, so its learning signal nearly vanishes — the vanishing gradient problem in deep stacks.
- **Tanh squashes into −1 to 1.** Similar to sigmoid but centred on zero, which usually trains a bit better. It still goes flat at the extremes, so it has the same fading problem.
- **ReLU is the modern default for hidden layers.** ReLU(x) = x if x is positive, else 0. It is simple, fast, and its signal does not fade for positive values — which is why deep networks finally became practical. Its flaw: a unit that goes negative stays at 0 and can stop learning forever — a "dead ReLU".
- **Leaky ReLU and GELU are fixes.** Leaky ReLU lets a tiny signal through for negative inputs (for example 0.01 × x), so dead units can recover. GELU is a smoother cousin used in modern language models.

- **Softmax turns final scores into probabilities.** For a classification output layer, softmax makes all class scores add up to 1, so the biggest one reads as "the model's confidence".
- **Building blocks beyond activations:** a layer = weights + activation. Common extras are **dropout** (randomly switching off units during training, so the network cannot rely on any single unit — a strong anti-overfitting trick) and **normalisation** (re-centring layer outputs so training stays stable).
- **An activation's slope is its learning signal.** During backpropagation, the signal a unit passes backwards is scaled by how steep its activation is at that point. A flat activation (sigmoid at the extremes) passes almost no signal — that is the vanishing gradient, seen from the unit's side.
- **Hidden layers and output layers have different jobs.** Hidden layers bend and build features, so they use ReLU-style activations. The output layer must answer the question asked — a probability, a yes/no, a number — so its activation is chosen for the answer format, not for speed.

Examples

Example 1 — Why straight lines are not enough

Try to separate two groups of points shaped like a ring: one group in the centre, one group around it. No single straight line can separate them — any line you draw cuts through one group or the other. A network with only linear units fails this task completely. Add ReLU bends after each layer and the network can fold its decision boundary into a circle. Same layers, same weights idea — the bend is the whole difference.

Example 2 — Reading the famous activations

| Input | Sigmoid | Tanh | ReLU | |------:|--------:|-----:|-----:| | −10 | ~0.00 | ~−1.00 | 0 | | 0 | 0.50 | 0.00 | 0 | | 10 | ~1.00 | ~1.00 | 10 |

Notice: sigmoid and tanh flatten at the extremes (their learning signal dies there), while ReLU keeps growing for positive inputs. That one difference is a big reason modern networks use ReLU (or its cousins) in hidden layers.

Example 3 — Picking activations for a small classifier

- **Hidden layers:** ReLU. Fast, simple, and it keeps the learning signal alive.
- **Output, yes/no question:** sigmoid — one number between 0 and 1, read as a probability.
- **Output, pick one of several classes:** softmax — probabilities that add up to 1.
- **If many ReLUs go dead during training:** switch to Leaky ReLU and see if the dead units recover.

Example 4 — Computing activations in code (with output)

```
def relu(x):
    return x if x > 0 else 0

def sigmoid(x):
    import math
    return 1 / (1 + math.exp(-x))

for x in [-2, 0, 3]:
    print(x, "relu =", relu(x), "sigmoid =", round(sigmoid(x), 3))
```

Output:

```
-2 relu = 0 sigmoid = 0.119
0 relu = 0 sigmoid = 0.5
3 relu = 3 sigmoid = 0.953
```

Read the output: ReLU kills the negative input completely (0) and passes the positive input unchanged (3). Sigmoid squeezes everything into 0–1, but notice how close 0.953 already is to 1 — push the input to 10 and it barely moves, which is exactly the flat, no-signal zone.

Example 5 — Diagnosing dead ReLUs

You inspect a trained network and find a hidden unit whose output is 0 for every input you try:

| Input to the unit | Output | |------------------:|-------:| | −5 | 0 | | 0 | 0 | | 4 | 0 | | 20 | 0 |

The unit is dead: its weights drifted so that its pre-activation sum is always negative, so ReLU always outputs 0, so no learning signal ever flows back through it. The fix to try first is Leaky ReLU (a small slope, such as 0.01 × x, for negative inputs), which lets a faint signal through and gives the unit a chance to recover. If many units are dead, the learning rate may also be too high — large jumps can kill units in bunches.

Example 6 — Real world: confidence percentages in a photo app

When a photo app says "cat 91%, dog 7%, bird 2%", that breakdown comes from a softmax output layer, sitting on top of ReLU hidden layers. The hidden layers do the feature-building work (edges → textures → parts), and softmax turns the final scores into probabilities that add up to 100%. If the app instead used ReLU on the output, you would get raw scores like 4.2, 0.8 and 0.3 — numbers that do not read as confidence at all. Choosing the output activation is choosing how the answer is presented to the user.

Quick recap

- Without activation functions, stacked layers collapse into one straight-line calculation.
- Sigmoid (0 to 1) and tanh (−1 to 1) squash outputs but go flat at extremes — vanishing gradients in deep networks.

- ReLU (0 for negative, x for positive) is the default hidden-layer choice; dead units are its flaw, Leaky ReLU is the common fix.
- Softmax turns final scores into probabilities for classification.
- Dropout and normalisation are the other everyday building blocks.

Common mistakes

1. **Using sigmoid in every hidden layer of a deep network** — the signal fades and early layers stop learning. Fix: use ReLU in hidden layers; keep sigmoid only for a yes/no output.
2. **Using ReLU on the output for probabilities** — ReLU outputs are not probabilities and do not add to 1. Fix: softmax for multi-class outputs, sigmoid for yes/no outputs.
3. **Forgetting dropout on a small dataset** — a big network memorises quickly. Fix: add dropout and watch the validation loss, not just the training loss.

1. **Leaving hidden layers with no activation at all** — a stack of plain weighted sums is still one straight-line calculation, however deep it looks. Fix: put an activation (usually ReLU) after every hidden layer.
2. **Using softmax for multi-label answers** — softmax forces one winner, so it is wrong when several answers can be true at once (a photo that is both "outdoor" and "contains a dog"). Fix: use one sigmoid per label, so each label gets its own independent yes/no probability.

Practice

1. Compute ReLU, sigmoid (roughly) and tanh (roughly) for inputs −2, 0 and 3. Which activation keeps a learning signal for the input 3?
2. A network must choose between "cat", "dog" and "bird". Which output activation fits, and why?
3. Half your hidden units output 0 for every input. Name the problem and the one change you would try first.

1. Compute ReLU and Leaky ReLU (slope 0.01 for negatives) for inputs −10, −1 and 5. Which one still passes a signal for −10, and how big is that signal?
2. A model must tag photos with any of these labels: "beach", "people", "food" — several can apply to one photo. Softmax or one sigmoid per label? Give one reason.
3. Your network's early layers barely change during training, but the last layers learn fast. Name the likely problem (think: DL.3's fading signal) and the activation change you would try first.

Practice quiz

1. Why does a network need activation functions?

- To make it faster
- Without a non-linear bend, stacked layers could only draw straight lines
- To save memory
- To print results

Answer: Without a non-linear bend, stacked layers could only draw straight lines

2. What is ReLU?

- A function that outputs x when x is positive and 0 otherwise
- A function that always outputs 1
- A type of layer
- A training method

Answer: A function that outputs x when x is positive and 0 otherwise

3. Which output activation fits 'pick one of cat, dog or bird'?

- ReLU
- Sigmoid
- Softmax — probabilities that add up to 1
- Tanh

Answer: Softmax — probabilities that add up to 1

Watch the video lesson: https://youtu.be/NP21Sd7tb9g

Read online: https://scopetreasure.com/notes/dl-activations-building-blocks · More free tools and notes at https://scopetreasure.com