FULCRUM COURSE NOTES · 04

Deep Learning: Foundations

Neural networks from the single neuron up to the transformer: backpropagation, what depth actually buys you, how attention replaced recurrence, and what happens inside a large language model.

FULCRUM

Utsav Poudel
Founder, Fulcrum

ABOUT FULCRUM

Who made this, and why it is free

Fulcrum is a volunteer-run non-profit. We help people from under-resourced regions learn artificial intelligence through one-on-one mentorship, research and publication supervision, project support, workshops, and a guest speaker series.

The premise is simple. Ability is spread evenly across the world; opportunity is not. What separates a capable student in Pokhara or rural Bihar from one at a well-funded lab is rarely intelligence. It is access to supervision, to reviewers, to equipment, and to the unwritten rules nobody writes down.

This deck is part of how we try to close that. It is free, it will stay free, and you may reuse and adapt it for your own teaching under CC BY 4.0. If it is useful to you, come and find us.

Find us

Website
fulcrumgo.github.io

More material
fulcrumgo.github.io/resources

Discord
discord.gg/gbQCGkupdJ

LinkedIn
linkedin.com/company/gofulcrum

Every deck we publish is free to download at fulcrumgo.github.io/resources

PART ONE

A neuron is a weighted sum and a squashing function

Everything else is arrangement and scale.

From neuron to network

- One unit computes `z = w·x + b`, then applies a non-linearity `a = σ(z)`.
- Stack units into a **layer**; stack layers into a network. Each layer's output is the next layer's input.
- Without the non-linearity, stacked linear layers collapse into a single linear layer, depth would buy nothing.
- The **universal approximation theorem** says one wide hidden layer can approximate any continuous function. It says nothing about whether you can find those weights.

Depth matters in practice because it lets the network reuse intermediate features, so it needs far fewer units than an equivalent shallow model.

Activation functions

- **Sigmoid** and **tanh** saturate at the extremes, where gradients vanish. This stalled deep networks for years.
- **ReLU** (`max(0, x)`) does not saturate for positive input, is cheap, and made deep training practical.
- **Leaky ReLU** and **GELU** address ReLU's dead-unit problem; GELU is standard in transformers.
- Output layer is chosen by the task: none for regression, sigmoid for binary, softmax for multi-class.

Backpropagation

Not a learning algorithm, an efficient way to compute gradients.

- The **forward pass** computes predictions and the loss, caching intermediate values.
- The **backward pass** applies the chain rule from the loss back to every parameter, reusing those cached values.
- Cost is roughly the same as the forward pass, which is what makes training large networks feasible at all.
- Frameworks build a computation graph and do this automatically, but understanding it is what lets you debug a model that will not learn.

Do this once

Implement a two-layer network and its gradients in NumPy, no framework. It takes an afternoon and permanently removes the mystery.

Why deep networks refuse to train

- **Vanishing gradients**, repeated multiplication by small numbers drives early-layer gradients to zero.
- **Exploding gradients**, the same mechanism in reverse; fix with gradient clipping.
- **Poor initialisation**, Xavier and He initialisation keep activation variance stable across layers.
- **Internal covariate shift**, layer input distributions move as earlier layers update, which normalisation addresses.

Normalisation and residuals

- **Batch normalisation** standardises activations per mini-batch. Enables higher learning rates, and behaves differently at train and test time, a frequent source of bugs.
- **Layer normalisation** normalises across features within one example, so it is independent of batch size. Standard in transformers.
- **Residual connections** (`out = F(x) + x`) give gradients a direct path backwards and made very deep networks trainable.
- Together, these three are why 2015-era networks could go deep when 2012-era ones could not.

The main architecture families

- **MLPs**, fully connected. Fine for tabular data; wasteful on images because they ignore spatial structure.
- **CNNs**, weight sharing and locality. Efficient wherever position matters and patterns repeat.
- **RNNs and LSTMs**, process sequences with a carried hidden state. Largely superseded, but still sensible for small-data sequence problems.
- **Transformers**, attention over a whole sequence at once, and now dominant across text, vision and audio. Part two takes these apart properly.

Training in practice

- **Learning rate** is the hyperparameter that matters most. Sweep it over orders of magnitude before tuning anything else.
- **Schedules**, warmup then cosine decay is a reliable default.
- **Batch size** trades gradient noise against throughput; larger batches usually need a larger learning rate.
- **Mixed precision** roughly halves memory and speeds training substantially on modern GPUs.

Debugging order: overfit a single batch. If the model cannot drive loss to near zero on ten examples, you have a bug, not a modelling problem.

Transfer learning

The single most useful technique if you do not have a large dataset or a cluster.

- Take a model pretrained on a large corpus; its early features are broadly reusable.
- **Feature extraction**, freeze the backbone, train a new head. Fast, and works with very little data.
- **Fine-tuning**, unfreeze some or all layers at a low learning rate. Better results, more data needed.
- **Parameter-efficient methods** tune a small number of extra weights rather than the whole model, which is what makes large models adaptable on modest hardware. Part three shows how.

Why this matters here

Nearly every Fulcrum mentee is working without institutional compute. Transfer learning and PEFT are what make serious work possible on a free Colab tier.

PART TWO

Attention, and the problem it was invented to solve

Everything so far applies to any network. This part is about the one architecture that now dominates the field.

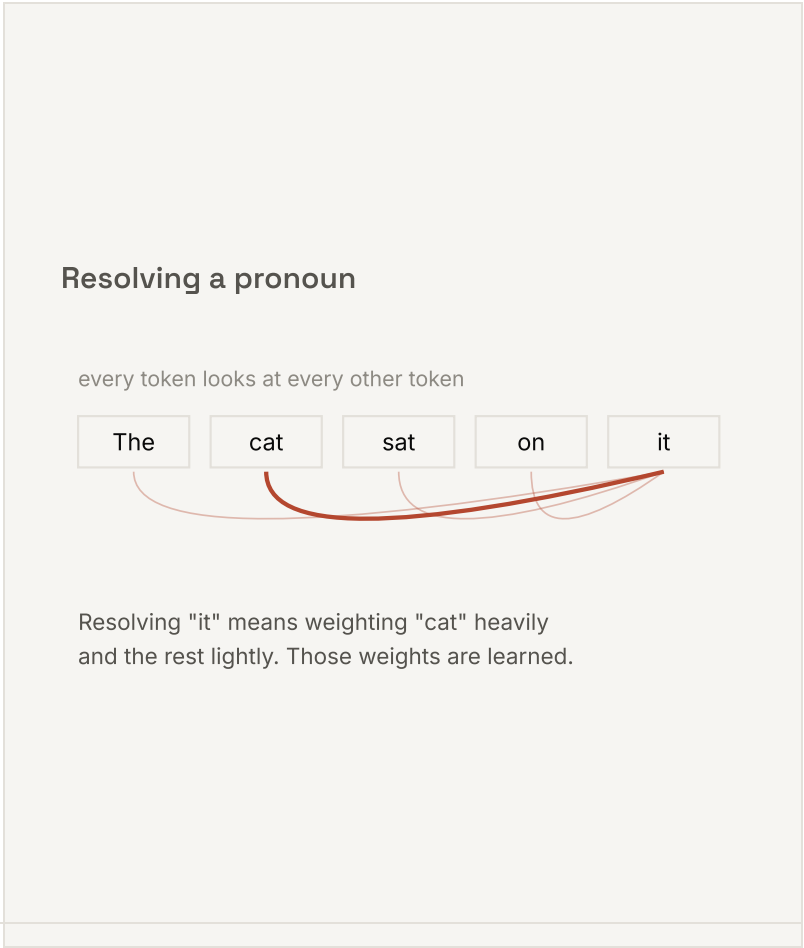
What was wrong with recurrence

- An RNN compresses everything seen so far into one fixed-size hidden state, a bottleneck for long inputs.
- It is **inherently sequential**, so it cannot exploit parallel hardware during training.
- Long-range dependencies degrade: gradients must travel through every intervening step.
- LSTMs and GRUs mitigated this with gating, but did not remove either limitation.

Attention, stated plainly

Let every position look directly at every other position, and learn how much to weight each.

- Each token produces three vectors: a **query**, a **key**, and a **value**.
- Compare one token's query against all keys to get a relevance score per token.
- Softmax those scores into weights; the output is the weighted sum of values.
- Nothing is compressed into a bottleneck, and every comparison happens in parallel.



The formula, decoded

`Attention(Q, K, V) = softmax(QKᵀ / √d) V`

- `QKᵀ`, every query dotted with every key, giving a score matrix of shape `n × n`.
- `/ √d`, scaling by the square root of head dimension. Without it, large dot products push softmax into saturation and gradients vanish.
- `softmax`, turns scores into weights that sum to one.
- `V`, the weighted sum of values is the output for that position.

That `n × n` matrix is the whole efficiency story: cost grows with the square of sequence length. Everything called 'efficient attention' is an attack on that term.

Multi-head attention

- Run several attention operations in parallel, each with its own learned projections.
- Different heads specialise. Some track syntax, some track coreference, some attend to position.
- Concatenate the heads' outputs and project back to the model dimension.
- Each head works in a smaller subspace, so total cost is comparable to one full-width head.

Head interpretations are suggestive, not definitive. Be careful reading too much meaning into attention maps. Attention weight is not the same thing as causal importance.

The rest of the block

- **Positional encoding**, attention is permutation-invariant, so position must be injected explicitly. Sinusoidal originally; rotary (RoPE) is now common.
- **Feed-forward network**, a per-position MLP, typically 4× the model width. This is where most parameters live.
- **Residual connections** around both sub-layers, plus **layer normalisation**.
- Stack this block N times. That is the entire architecture.

PART THREE

From architecture to language model

The architecture is simple. The behaviour comes from scale and training objective.

Encoder, decoder, or both

- **Encoder-only** (BERT), bidirectional context, trained by masking tokens. Good for classification and retrieval.
- **Decoder-only** (GPT family), causal masking so each position sees only the past. Trained to predict the next token. Good for generation.
- **Encoder-decoder** (T5, original Transformer), natural fit for translation and summarisation.
- Decoder-only has become dominant largely because next-token prediction scales so cleanly.

Tokenisation

- Models operate on **subword tokens**, not characters or words, via byte-pair encoding or similar.
- This keeps vocabulary manageable while handling unseen words by composition.
- Tokenisers are trained mostly on English-dominant corpora, so other scripts often fragment into far more tokens.
- That is a real equity issue: the same sentence in Nepali can cost several times more tokens than in English, making it slower and more expensive to process.

Worth checking yourself

Run a sentence in your own language through a tokeniser and count. It is a concrete, publishable observation about who these systems are cheap for.

How training actually proceeds

- **Pretraining**, next-token prediction over an enormous corpus. This is where almost all compute goes.
- **Supervised fine-tuning**, training on curated instruction-and-response pairs to make the model follow requests.
- **Preference tuning** (RLHF, DPO), optimising against human preference comparisons.
- The base model has the knowledge; the later stages shape how it behaves.

What these models do and do not do

- They model the **distribution of text**. Fluency is the objective; truth is not.
- **Hallucination** is not a bug bolted on top. It is the same mechanism producing a plausible continuation that happens to be false.
- **Context window** bounds what the model can attend to. Beyond it, information is simply gone.
- They are strong at transformation (summarising, rewriting, translating) and much weaker as a factual store.

Useful framing for teaching: it is an extremely good pattern completer. That explains both the impressive results and the confident errors.

Working with these models on modest hardware

- **Quantisation** to 4-bit or 8-bit lets sizeable open models run on a single consumer GPU.
- **LoRA** trains small low-rank updates instead of full weights, fine-tuning becomes affordable.
- **Retrieval-augmented generation** grounds answers in documents you supply, which is usually better than fine-tuning facts in.
- **Prompting well** costs nothing and is frequently the highest-return intervention available.

Reading and reproducing papers

- Read in this order: abstract, figures, results, then method. Introductions are the least informative part.
- Ask what the **baseline** is and whether the comparison is fair, same data, same compute, same tuning effort.
- Check whether code and weights are released. Unreleased results should be treated as provisional.
- Reproducing a paper is a legitimate contribution, and it teaches more than reading twenty.

Open questions worth a paper

- Tokenisation and cost equity across scripts and low-resource languages.
- Evaluation beyond English. Most benchmarks do not transfer, and building one for your language is a genuine contribution.
- Small-model performance under tight compute, which matters far more outside well-funded labs.
- Careful failure analysis of deployed systems in specific domains. Unglamorous, useful, and publishable.

Choosing a question

The frontier is crowded and expensive. The gaps are in places large labs have no reason to look, which is precisely where our mentees live.

ABOUT FULCRUM

Bring us something you are building

Fulcrum mentors work through exactly this, from a first network to a research question worth submitting. Free, and open to anyone from an under-resourced region.

Fulcrum is a volunteer-run non-profit helping people from under-resourced regions learn AI through free mentorship, research supervision, and open material like this deck. Founded 2025 in Kathmandu, Nepal. Nothing we do costs anything.

Website
fulcrumgo.github.io

Every deck, free
fulcrumgo.github.io/resources

Apply for mentorship
discord.gg/gbQCGkupdJ

Partnerships
linkedin.com/company/gofulcrum