Introduction to Neural Network Interpretability

Part I: From Random Machines to Trained Machines

Contact Information
For permission requests or inquiries, please contact: ruchirtewari@gmail.com

Introduction to Neural Network Interpretability

Part I: From Random Machines to Trained Machines

Chapters 1–4

Ruchir Tewari

August 31, 2026

How to Read This Book

This is the first half of a book that ends at the frontier of neural network interpretability. The destination - understanding and controlling what happens inside trained models (Part II, Chapters 5–8) - requires foundations that are usually scattered across information theory, differential geometry, and optimization. Part I collects them into a single arc:

1. **Chapter 1 - Statistical Foundations.** Where does structure in generated text come from? (Answer: not from randomness but from conditioning - the line from Markov's letter statistics through Shannon's entropy to the modern language-modeling objective.)
2. **Chapter 2 - Geometric Foundations.** How far apart are two probability distributions, honestly? (Answer: Fisher information turns distinguishability into curvature, and the Fisher–Rao metric turns families of distributions into a landscape with true distances.)
3. **Chapter 3 - Neural Networks as Probability Distribution Machines.** What, mathematically, does a trained network do? (Answer: it shapes probability distributions - sequentially by conditioning in transformers, globally by warping in flow and diffusion models, and in parallel by splitting the job across heads and experts.)
4. **Chapter 4 - Training Dynamics.** How does the machine get tuned? (Answer: by descending a loss landscape whose true geometry is the Fisher metric - explicitly in natural-gradient methods, implicitly in the flat-minima bias of plain stochastic gradient descent.)

The two threads - statistical (Chapters 1, 3) and geometric (Chapters 2, 4) - braid together deliberately. Chapter 1 explains *what* a generative model must produce; Chapter 2 explains how to *measure* progress toward it; Chapter 3 shows the machines that produce it; Chapter 4 shows how measurement geometry governs the tuning of those machines. Part II then opens the trained machine and asks what formed inside.

Some conventions, shared with Part II:

- Citations appear inline as (Author, Year); full references, with the filenames of archived PDFs in `new_book/references/`, are collected in the References chapter at the end.
- Short **Neuroscience parallel** paragraphs appear inline where an idea has a direct counterpart in the study of biological brains - stated concretely: what the object is in the brain, and what the corresponding object is in the artificial network.
- Chapters close with three collected sections: **Takeaways**, **Research Directions**, and **Programs** - hands-on projects that let you reproduce the chapter's main claims yourself. All programs run on a laptop with Python 3, `numpy`, `matplotlib`, and (for Chapters 3–4) `torch`.
- No result is presented without saying what kind of evidence supports it. In Part I the mathematics is mostly settled; the places where claims are empirical rather than proved (notably in Chapter 4) are flagged explicitly.

Contents

1. Statistical Foundations

In *One Two Three… Infinity* (1947), George Gamow supposes a printing press that prints letters automatically one after another which given enough time would produce every line from Shakespeare. In his construction, the letter producing dials move deterministically as a car meter dial turns the next dial when completing a turn. To produce all possible character combinations of an 80 character line would require printing $27^{80} = 10^{114}$ combinations. If instead the machine were built to output randomly on each line a sequence of 10 words from Shakespeare's vocabulary of about 15000, there would be a smaller number of 15000^{10} or 10^{42} combinations. If the machine were to write a line a billion times a second for a billion years, it would produce only 10^{26} lines and would be far from writing every possible line with either character or word combinations.

Now suppose the machine were driven by a random number generator with a controlled probability distribution of the production of the words. The probability distribution could model the frequency of the words in the language, the frequency of two or three word combinations, and be able to learn patterns in Shakespeare according to a probability distribution. The output by the machine would significantly improve on the odds of producing text that looked like Shakespeare, yet produce gibberish text most of the time. The word *model* is used here as a synonym for a probability distribution. A modern language model is an answer to the question the text producing machine raises: what has to be added to randomness before it starts producing language?

Structure of this chapter. Section 1.1 asks what separates a machine that emits letters from a machine that emits language, and finds the founding observation (Markov's 1913 letter counts): text carries measurable local structure. Section 1.2 makes conditioning precise - the Markov property, the ladder of ever-richer n-gram approximations, and the state-explosion wall where counting runs out. Section 1.3 supplies the measures - surprise, entropy, cross-entropy, perplexity - that turn "closer to language" into a number, the number every modern language model is trained to minimize.

1.1 The Statistical Machine

The printing press samples from a probability distribution that carries *zero information about language*. Improving the press means changing the distribution - concentrating its mass on the vanishingly small subspace of strings that look like language.

Markov reads Pushkin. The Russian mathematician Andrei Markov in 1913 took the first 20,000 letters of Pushkin's verse novel and counted, by hand, how often a vowel was followed by a vowel, a vowel by a consonant, a consonant by a vowel, and a consonant by a consonant (Markov, 1913). The counts were dramatically non-uniform: a vowel followed a consonant far more often than it followed another vowel. Letters in real text are not independent draws - each letter carries statistical information about its neighbor.

Markov's motivation was a technical dispute about whether the law of large numbers applies to dependent events. *Eugene Onegin* was a convenient sequence of dependent events. The

byproduct was the founding observation of statistical language modeling. Language has *local statistical structure*. That structure is *measurable by counting*. And a machine equipped with those counts is no longer searching a flat space - it searches a space already shaped like the target. Markov showed letters are not independent in their statistics and one does not need the entire history to make a prediction about the next letter.

1.2 The Markov Property

1.2.1 Memorylessness, made precise

A stochastic process has the **Markov property** when its next state depends only on its current state, not on the path that led there:

$$P(X_{t+1} \mid X_t, X_{t-1}, \ldots, X_1) \;=\; P(X_{t+1} \mid X_t).$$

The word "memoryless" is standard but slightly misleading: the process does have memory - one state's worth. The design question that drives the rest of this chapter is: *what should the state be?* Markov's Pushkin analysis used the smallest interesting state, the class (vowel/consonant) of the current letter. Enriching the state is the single move that separates every generation of language model from the previous one.

An order-k (or "n-gram") model takes the state to be the last k symbols: the next character is drawn from the empirical distribution of what followed those k characters in a training corpus. An order-0 model draws each symbol independently from the corpus letter frequencies.

1.2.2 Climbing the ladder

Shannon (1948) made the ladder vivid by generating text from successively richer approximations to English, and the samples remain the fastest way to feel what each increment of context buys:

- **Zero-order** (the uniform distribution; Shannon's naming starts here): *XFOML RXKHRJFFJUJ ZLPWCFWKCYJ…* - noise.
- **First-order** (letter frequencies only): *OCRO HLI RGWR NMIELWIS EU LL…* - the texture changes; vowels appear at roughly English rates, but nothing is pronounceable.
- **Second-order** (letter pairs, Markov's statistics): *ON IE ANTSOUTINYS ARE T INCTORE ST…* - pronounceable fragments emerge; word-like units appear.
- **Third-order** (letter triples): *IN NO IST LAT WHEY CRATICT FROURE…* - most units could be English words; some are.
- **First-order word model**: *REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME…* - real words, no grammar.
- **Second-order word model**: *THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS…* - locally grammatical stretches several words long.

Each rung conditions on more context, and each rung's output is recognizably closer to language. An interactive companion to this ladder is at `html/sentence-generators.html`: uniform and Markov sentence generators side by side, with dials for the conditioning, so you can watch the samples change as structure is added.

1.2.3 The state-explosion wall

The ladder cannot be climbed indefinitely. An order-k character model over a 27-symbol alphabet has 27^k possible states. An order-k *word* model over a 50,000-word vocabulary has $50{,}000^k$. At $k = 5$ words, that is 3×10^{23} contexts. Almost none of them ever occur, even in an internet-scale corpus. A context that never occurs contributes no counts, and a next-word distribution cannot be estimated from zero counts. Conditioning on long contexts by counting is therefore statistically impossible: every long context is new. Call this the **state-explosion wall**. It is not unique to language modeling. Other fields hit the state-explosion wall and the ways they got past it serve as directional hints, even where the full mechanism is still not understood.

- **Statistical mechanics.** A gram of gas contains $\sim 10^{22}$ molecules. Its space of microstates dwarfs $50{,}000^5$ beyond comparison. Boltzmann and Gibbs got past the explosion by refusing to track microstates at all. They tracked a handful of macroscopic variables (temperature, pressure, entropy) that summarize everything about the microstate that matters for prediction.
- **Data compression.** A compressed file is short for one reason: the source is not uniform. A 27-symbol alphabet costs 4.75 bits per character if coded naively; because English's statistics are lopsided, about 1 bit per character suffices. Shannon's source-coding theorem says compressibility comes from probability being extremely nonuniform over possibility space and this is exploited in compression algorithms such as Huffman coding which gives short bit strings to common symbols and longer bit strings to rare symbols, and converts probability into code length.
- **Biology.** An oak tree contains on the order of 10^{12} cells, arranged in structures no blueprint could enumerate. The acorn stores none of that arrangement. It stores a compact generative program (a genome of a few gigabits) plus a developmental process that rebuilds the structure on demand. The escape move: store a *generator* of the structure, not the structure itself.

The repeated solution: when states explode, find the compact object (sufficient statistics, the typical set, a generative program) that produces the needed predictions without a table of states. Language modeling after the n-gram era reached for the same solution. Its compact object is a *learned state*: a vector-valued summary of the context, produced by a trained network, that generalizes across contexts instead of counting them.

1.3 Measures over Outcomes and Distributions

1.3.1 Surprise and its expectation, entropy

Shannon's 1948 paper built on the ladder samples to define the quantity that measures what the ladder is climbing toward. Define the **information content** or **surprise** of an outcome x with probability $p(x)$ as $-\log_2 p(x)$, measured in **bits**: an outcome with probability $\frac{1}{2}$ carries 1 bit; probability $\frac{1}{1024}$ carries 10 bits. Surprise is a measure for a single outcome. Consider two outcomes of a weather forecast. The forecast said 90% sun, and it was sunny: the surprise is $-\log_2 0.9 = 0.15$ bits - expected, barely registered. The forecast said 90% sun, but it rained: $-\log_2 0.1 = 3.32$ bits of surprise - a shock.

Entropy is the probability weighted average of the bits required for each outcome in a distribution:

$$H(p) \;=\; -\sum_{x} p(x)\, \log_2 p(x). \tag{1.1}$$

It is a measure over a distribution, not a single event: the bits required for each outcome, weighted by that outcome's probability and summed - the average bits needed to encode an outcome drawn from the distribution. It is the expectation of the surprise, $\mathbb{E}$[surprise].

For a given number of outcomes, entropy is maximized by the uniform distribution and shrinks as probability mass concentrates. For a distribution with n outcomes each of probability $1/n$, entropy is $n \cdot (1/n) \cdot \log_2 n = \log_2 n$. This increases with n, which makes entropy a measure of spread: among equiprobable sets of outcomes, the larger the set, the higher the entropy. It is a rigorous measure of "how unpredictable the next symbol is". When measured in base 2 logarithm, it has the unit of bits, and using natural logarithms, it has the unit of nats.

Each next rung of the ladder conditions on more context and therefore faces less uncertainty per character.

1.3.2 Cross-entropy and perplexity

A code is a rule that converts each outcome into a sequence of bits. Now a code can be efficient for a distribution if it produces bit lengths that match the information content of the outcome derived from their probability. In general this is not true - let's take an example. Say we have 4 bit codes 0000, 0001 to 1111 that encode 16 outcomes that are not equally probable and that are in fact highly skewed. Say 4 outcomes have 22% probability each for a total of 88% and the remaining 12 outcomes have a 1% probability each. The entropy calculation shows that this requires 2.37 bits on average, but we are paying 4 bits per outcome. This motivates a measure called cross-entropy.

Cross-entropy is a measure of the difference between two probability distributions. A model that produces probabilities according to a distribution q trying to predict text that is actually distributed as p pays the following number of bits on average:

$$H(p, q) \;=\; -\sum_{x} p(x) \log_2 q(x) \tag{1.2}$$

Note this is not symmetric.

If the two distributions p and q are equal, cross-entropy reduces to the entropy of q (and not to zero).

Perplexity is the exponential of cross-entropy, $2^{H(p,q)}$: the effective number of equally likely choices the model faces per symbol.

The difference between the cross-entropy and the entropy of the target distribution p, is the divergence of p with respect to q, and this goes to zero as q approaches p. Minimizing cross-entropy on training text is the objective by which GPT-class models are trained and the objective is reached when the divergence is zero or minimum - training consists of descending that scale by reshaping q toward p. Subtracting equation 1.1 from equation 1.2 we get another measure of the difference between two distributions:

$$H(p, q) - H(p) = D\,(p \,\|\, q) \tag{1.3}$$

This divergence is not symmetric either. There are other measures of distances between probability distributions that are symmetric such as JS divergence that are discussed later.

The companion page (`html/sentence-generators.html`) scores each of its generators by cross-entropy against Shakespeare, so the entropy ladder and the perplexity numbers of this section can be explored by hand.

A neural network can be thought of as repeatedly transforming a broad probability space into highly concentrated internal distributions. The transformed objects are internal representations, which this book builds toward understanding.

Neuroscience parallel. Sensory neurons face the entropy problem too. Barlow (1961) proposed the **efficient coding hypothesis**: early sensory neurons recode their inputs to remove statistical redundancy. Spikes are metabolically expensive and channel capacity is limited, so a neuron should spend its signaling on the surprising part of the input, not the predictable part. Laughlin (1981) confirmed a sharp version of this prediction in the blowfly eye. He measured the contrast-response curve of a specific visual interneuron and compared it to the statistics of natural scenes. The neuron's response curve follows the cumulative distribution of contrasts found in nature, allocating its limited output range where the input probability mass actually is.

Takeaways

- Uniform randomness never produces language; conditioning does. Markov's 1913 hand count of Pushkin showed that letters carry measurable local structure, and a machine storing those counts already searches a space shaped like the target.
- The design question of every language model is *what the state is*. The ladder from order-0 letter models to word-pair models enriches the state rung by rung - and ends at the state-explosion wall, where every long context is new and counting cannot estimate it.
- The escape, seen in statistical mechanics, compression, and biology alike, is to replace the table of states with a compact generator. For language that generator is a *learned state* - the vector a trained network computes - which is where Chapter 3 picks up.
- Surprise, entropy, cross-entropy, and perplexity turn "closer to language" into a number. Cross-entropy against data is that number: the objective every GPT-class model is trained to minimize, and the bridge from this chapter's statistics to Chapter 2's geometry.

Research Directions

From §1.2: Where does the entropy curve flatten, per domain?

Measure conditional entropy $H(X_t \mid X_{t-k}, \ldots, X_{t-1})$ on matched-size corpora of prose, code, mathematics, and dialogue. Each domain's curve has a knee where local statistics stop helping. Does the knee's position predict which domains benefit most from long-context architectures? The measurement is pure counting - no training - and it maps where in the input distribution the learned state of Chapter 3 pays off.

From §1.3: What does the tokenizer buy, in bits?

Train order-4 models on one corpus tokenized three ways - characters, words, byte-pair subwords (Section 3.1 defines the third) - and convert all cross-entropies to bits per character. The raw numbers differ widely across tokenizations while measuring the same underlying uncertainty; the conversion exposes how much of a modern model's apparent advantage is the representation of text rather than knowledge of it. A careful version of this comparison is surprisingly rare in print.

Programs for Chapter 1

`uniformrandom_vs_markov_text.py` Download a public-domain text (Project Gutenberg Shakespeare works). Generate 500 characters each from: uniform random; order-0 (letter frequencies); order-1 through order-4 character Markov models built by counting; order-1 and order-2 word models. Print the samples as a ladder, and for each, report per-character cross-entropy against held-out text. You will reproduce Shannon's 1948 table on your own corpus and watch language condense out of noise, one order at a time. *Type: script, prints ladder + entropy table. Deps: numpy only.*

`entropy_vs_context_length.py` Using the same corpus, estimate conditional entropy $H(X_t \mid X_{t-k}, \ldots, X_{t-1})$ for $k = 0 \ldots 8$ characters by counting (with add-one smoothing), and plot the falling curve. Mark Shannon's reference points (4.75, ∼4.1, ∼3.3, ∼1.0 bits). Then show the estimator break: plot the number of distinct contexts observed vs. 27^k - the state-explosion wall of §1.2 appearing as data sparsity in your own counts. *Type: script + matplotlib. Deps: numpy.*

2. Geometric Foundations

We want to adjust the probability distributions that are generated to get them closer to a target. Entropy tells you how uncertain a machine's next symbol is, but does not give a way to control it. To do this we add *dials* to the machine, that change how the output distribution responds when you nudge them. Training is dial-nudging a few million or billion knobs at a time. This chapter builds the mathematics of dial-sensitivity. It is a second notion of "information," due to Fisher and Rao, distinct from the Markov–Shannon one.

Here's a picture to hold, extending Gamow's typing machine: imagine a whole *colony* of letter machines, each with slightly different dial settings - one biased 40% toward vowels, another 41%, another 60%. Two machines whose outputs are nearly indistinguishable should count as "close," whatever their dial labels say; two machines with obviously different output should count as "far," even if their dials differ by a hair. The mathematics of this chapter is the construction of that honest ruler.

Structure of this chapter. Section 2.1 defines **Fisher information** through the most concrete picture available: nudge a parameter by δ and ask how distinguishable the new output distribution is from the old one; the answer is a bowl-shaped curve whose curvature at the bottom *is* the Fisher information. Section 2.2 carries the construction into trained networks: the score decomposes through the network's Jacobian, two dial examples are computable by hand, the Cramér–Rao bound prices every readout, and Monte Carlo with Fisher–vector products make the matrix usable at billion-parameter scale. Section 2.3 assembles the bowls (one at every parameter setting) into the **Fisher–Rao metric**, a Riemannian geometry on the space of distributions, with geodesics as true shortest paths and a uniqueness theorem explaining why this metric and no other. Section 2.4 sets Shannon and Fisher side by side and shows they measure different things, uncertainty of *outcomes* versus estimability of *parameters*, which can point in opposite directions for the same distribution. Chapter 4 will put this geometry to work in training.

2.1 Fisher Information as Curvature

2.1.1 The ruler first: Kullback–Leibler divergence, defined

Let's look at divergence again. For two distributions p and q over the same outcomes, define the KL divergence as

$$D_{\mathrm{KL}}(p \,\|\, q) \;=\; \sum_{x} p(x) \log \frac{p(x)}{q(x)} \tag{2.1}$$

It follows from 1.3 and is the average, *taken under* p, of the log-ratio between the two probability assignments. The double-bar notation "$\|$" is a separator, not an operation: it marks which distribution plays which role. The distribution on the *left* of the bar is the reference - the one assumed to actually generate the data, the one the expectation averages over; the distribution on the *right* is the one being judged against it. Read $D_{\mathrm{KL}}(p \,\|\, q)$ as "the divergence *of* q *from*

p," or operationally: the expected number of extra bits paid per outcome when outcomes truly drawn from p are encoded using a code built for q.

The order matters because the two roles are not interchangeable: $D_{\mathrm{KL}}(p \,\|\, q) \neq D_{\mathrm{KL}}(q \,\|\, p)$ in general and the divergence is not symmetric. Additionally it violates the triangle inequality, and is therefore *not a distance*. The two orderings ask different questions - "how surprising is q's code to a p-world?" versus the how surprising p's code is to a q world .

However, for *infinitesimally* separated distributions the asymmetry vanishes, and a symmetric measure emerges from an asymmetric one.

2.1.2 From surprise to the score to Fisher information

We defined the surprise of an outcome x as $-\log p_\theta(x)$: how shocked the machine is to see x. Written with the sign flipped, $\log p_\theta(x)$ is the **log-likelihood** of x under the model.

Now ask how the shock responds to the dials. Take a parametrized family p_θ (the vowel-bias dial on a letter machine, the many weights in a transformer) and differentiate the log-likelihood with respect to θ. This gradient of the surprise with respect to the model weight is called the **score**, s(x):

$$s(x) \;=\; \nabla_\theta \log p_\theta(x),$$

It is a vector in the parameter space, and points in the direction that makes x more likely - the direction θ would move if x alone were the training set. Different outcomes x vote for different directions.

The log goes on *before* the gradient for two structural reasons. First, products become sums: a sequence factorizes as $p_\theta(x_{1:T}) = \prod_t p_\theta(x_t \mid x_{<t})$, and the log turns the product into a sum whose gradient is just the sum of per-token scores. Second, the log makes the gradient measure *relative* change: $\nabla_\theta \log p = (\nabla_\theta p)/p$, the raw sensitivity normalized by the current probability. What matters for information is not how much p moves absolutely but how much it moves relative to where it stands.

Averaged over the model's *own* outcomes, the votes cancel, or the expectation of the score is zero.

$$\mathbb{E}_{p_\theta}[s] \;=\; \sum_x p_\theta(x)\, \nabla_\theta \log p_\theta(x) \;=\; \nabla_\theta \sum_x p_\theta(x) \;=\; \nabla_\theta 1 \;=\; 0.$$

This is because a machine that fed the data it expects asks for no parameter change. Each score may be non-zero but the average score is zero as the model matches reality - if it did not, this would be the training gradient.

While mean is zero, we can still look at the variance - there is information in the spread. Define the covariance of the score as follows:

$$F(\theta) \;=\; \mathbb{E}_{x\sim p_\theta}\Big[\, s(x)\, s(x)^\top \Big],$$

This covariance of the score is called the **Fisher information matrix**. Read the covariance as disagreement among outcomes. Large F: different observations push θ hard in different directions - each outcome reveals something distinct about the dials, and the output is informative about the machine that produced it. Small F: every observation pushes weakly or in the same direction - moving θ barely changes which outcomes are likely, and the dial hides. Fisher information is how much a single observation can tell you about the parameters, or equivalently how sensitive the model's predictions are to parameter changes, averaged over what the model itself expects to see.

The divergence arrives at the same matrix. Nudge the parameter from θ to $\theta+\delta$ and measure distinguishability with the divergence of the previous subsection, $g(\delta) = D_{\mathrm{KL}}(p_\theta \,\|\, p_{\theta+\delta})$, Taylor-expanded around $\delta = 0$: $g(\delta) \approx g(0) + g'(0)\,\delta + \frac{1}{2}\,g''(0)\,\delta^2$. The first two terms vanish. The zeroth-order term is $g(0) = D_{\mathrm{KL}}(p_\theta \| p_\theta) = 0$; the first-order term vanishes because $\delta = 0$ is a minimum, and the slope there is zero - the same fact as the expected score being zero, in geometric form. The leading survivor is the quadratic term, with Taylor's standard $\frac{1}{2}g''(0)$ coefficient, and $g''(0)$ is the covariance just built. This is the central formula of the chapter:

$$D_{\mathrm{KL}}(p_\theta \,\|\, p_{\theta+\delta}) \;\approx\; \tfrac{1}{2}\,\delta^\top F(\theta)\,\delta.$$

One object, two constructions: from the inside, F is the covariance of per-observation parameter votes; from the outside, it is the curvature of distinguishability under a nudge. The two constructions agree because both are second derivatives of the same underlying surprise.

Notice also what the expansion delivered: $\delta^\top F \delta$ is symmetric in $\delta \to -\delta$, and expanding the KL divergence in the *other* order, $D_{\mathrm{KL}}(p_{\theta+\delta} \| p_\theta)$, gives the *same* quadratic form with the same F. The asymmetry of KL is a third-order effect. Locally - and only locally - distinguishability is symmetric, which is exactly what a metric requires and why Section 2.3 can build one.

2.1.3 Which bowl? Getting the picture right

The quadratic $\frac{1}{2}\delta^\top F \delta$ is a bowl, and the bowl image recurs so let's be precise now about which bowl this is. On the horizontal axes lies the *nudge* δ to the parameters at their current value - not the parameters themselves, and not their target distribution p. On the vertical axis, the divergence between the original machine and the nudged one. The bottom of this bowl sits at $\delta = 0$ with height exactly zero, *always, by construction, at every parameter setting* θ - at a well-trained optimum, at a terrible initialization, anywhere. There is one such bowl at every point of parameter space, and its shape (not its location) holds information: steep narrow walls mean a hair-trigger dial whose tiniest turn produces detectably different output (high Fisher information); wide shallow walls mean a loose dial you can turn without anyone noticing (low Fisher information). It is a *curvature*, not a slope - the slope at the bottom is zero by construction.

This is *not* the loss bowl of stochastic gradient descent. The training-loss surface is a function of the parameters themselves, with the data held fixed; its minimum sits wherever training happens to converge, its depth and slope carry meaning, and there is one landscape for the whole problem. The KL bowl is a function of a *hypothetical nudge*, re-centered at whatever θ you currently occupy, and its minimum is always trivially zero. The two are related - for log-loss, the expected Hessian of the loss under the model's own distribution equals F, which is why the pictures rhyme and why Chapter 4 can use F as a stand-in for loss curvature - but confusing them costs you the key property: the loss bowl tells you *how wrong* you are; the KL bowl tells you *how sensitive* you are, and sensitivity is defined even where wrongness isn't.

Is a sharp bowl good or bad? The question has no fixed answer. Fisher information is *descriptive*, a sensitivity, and whether sharp is good depends on who is asking. For an *estimator* trying to identify the dial from output, sharp is good - the parameter is pinned precisely (Section 2.2 works this out). For *robustness*, sharp is bad - a hair-trigger direction is one where tiny parameter noise, quantization error, or a fine-tuning step visibly changes behavior. For an *optimizer*, sharp means "step carefully": Chapter 4's natural gradient divides by F precisely to take small steps in sharp directions and bold steps in flat ones. Same number, three readings.

2.1.4 Whose probabilities? What the network "knows"

A network is a pile of matrix multiplications; so in what sense does it *have* a p_θ for KL to compare - how does it know these numbers are probabilities ? The answer is: by construction, at the output. A language model's final layer produces a vector of real numbers (logits), and the **softmax** function exponentiates and normalizes them into nonnegative numbers summing to one - a probability distribution over the vocabulary, whether or not anything "believes" it. The training objective then makes the label stick: cross-entropy loss rewards exactly the calibration of those numbers against observed next tokens, so a trained model's outputs are not merely normalized but *meaningfully* probabilistic - they are the quantities the objective sharpened. For a whole sequence the model assigns $p_\theta(x_{1:T}) = \prod_t p_\theta(x_t \mid x_{<t})$, each factor a softmax output. The network knows nothing; the architecture guarantees normalization, and the objective supplies the semantics. $D_{\mathrm{KL}}(p_\theta \| p_{\theta+\delta})$ is then a statement about the two machines' output distributions - the observable channel - not about any hidden interpretation.

This resolves a question the formula quietly raises: F is defined by an expectation over $x \sim p_\theta$ - outcomes drawn from *the model itself*, not from training data. Fisher information is a property of the machine at its current setting, measurable in principle by letting the machine run and watching how its own outputs vote.

2.2 Fisher Information in Trained Networks

2.2.1 The score and the network Jacobian

For the letter machine, $\nabla_\theta \log p_\theta(x)$ is one derivative. For a transformer it decomposes, and the decomposition is where the geometry meets the architecture. Write the model as parameters $\theta \mapsto$ logits $z(\theta) \mapsto$ probabilities $p = \mathrm{softmax}(z)$. By the chain rule, the score is

$$s(x) \;=\; \nabla_\theta \log p_\theta(x) \;=\; J_z^\top \, \nabla_z \log p(x),$$

where $J_z = \partial z / \partial \theta$ is the **Jacobian** of the logits with respect to the parameters - the object backpropagation computes. Substituting into the covariance form,

$$F(\theta) \;=\; \mathbb{E}\Big[J_z^\top \, G(z) \, J_z \Big], \qquad G(z) = \mathrm{diag}(p) - p\,p^\top,$$

the Fisher matrix of the softmax itself, conjugated by the network Jacobian. Read it as a *pullback*: the output simplex carries a fixed, known information geometry (G), and the network's Jacobian pulls that geometry back onto the weight space - each parameter direction inherits exactly as much curvature as it can move the output distribution. Directions the Jacobian kills (parameter changes the output not at all) get zero Fisher information; directions the Jacobian amplifies get more. This form - known output geometry composed with a learned Jacobian - is also the computational workhorse: it is the structure that the approximations below exploit.

2.2.2 Two dials you can compute by hand

The biased coin. For a Bernoulli distribution with heads probability p,

$$F(p) \;=\; \frac{1}{p(1-p)}.$$

At $p = 0.5$ the Fisher information is 4; as $p \to 0$ or 1 it diverges. A fair coin is the *loosest* dial (nudging 0.50 to 0.51 changes the output stream imperceptibly), while near-deterministic coins are hair-triggers: at $p = 0.99$, the same 0.01 nudge to 1.00 eliminates tails entirely, an unmistakable change. Distinguishability per unit of dial is not constant across the dial's range, and that non-constancy is the whole reason a geometry is needed.

The Gaussian mean. For a Gaussian with known standard deviation σ, the information about the mean is $F(\mu) = 1/\sigma^2$. A narrow distribution pins its mean sharply - every sample is a precise vote; a wide one leaves the mean blurry. Hold this example: it returns in Section 2.4 as the cleanest demonstration that Fisher and Shannon disagree.

2.2.3 Why estimators care: the Cramér–Rao bound

Fisher information earns its name from estimation theory. The **Cramér–Rao bound** states that any unbiased estimator $\hat{\theta}$ built from n samples has variance

$$\mathrm{Var}(\hat{\theta}) \;\geq\; \frac{1}{n\,F(\theta)}.$$

The bowl's curvature sets the best achievable precision for pinning down the dial from output alone. This is the lie-detector reading of the same mathematics. You watch output from one of two secretly different machines. Your ability to tell which machine you face is governed by F. High curvature: the output betrays its parameter. Low curvature: the parameter hides. The bound applies to any reader whatsoever - human analyst, statistical test, or trained classifier. Any attempt to decode a hidden quantity from a system's observable behavior operates under this ceiling, whatever the decoder.

2.2.4 Measuring Fisher information in a billion-dial machine

For the coin and the Gaussian, F came out in closed form. For a transformer there is no analytic formula - the expectation runs over the model's own output distribution through a billion-parameter Jacobian - and worse, the full matrix could not even be *stored*: at d parameters it has d^2 entries, which for $d \sim 10^9$ is 10^{18} numbers. Fisher information for networks is therefore always *estimated* and always *structured*. The standard moves, in increasing order of approximation:

- **Monte Carlo, exactly.** The covariance form is an expectation over $x \sim p_\theta$, so sample: run the model, draw outputs from its own distribution, backpropagate $\log p_\theta(x)$ to get one score vector per sample, and average outer products. This is unbiased for the true F at any sample budget - the estimate is noisy, never wrong on average.
- **Fisher–vector products.** Most uses need Fv for chosen directions v, not F itself - and Fv is computable by automatic differentiation without ever materializing the matrix, the same trick that makes Hessian-vector products cheap. Storage objection dissolved.

- **The empirical Fisher.** Replace samples from the model with training examples and their labels - averaging outer products of the loss gradients you were computing anyway. Nearly free, widely used, and subtly *different*: at a poorly fit point the empirical and true Fisher can disagree badly, a distinction that matters in practice and is often ignored.
- **Structured approximations.** Keep only the diagonal (one sensitivity per parameter), or per-layer Kronecker-factored blocks (K-FAC), trading fidelity for tractability.

What is the calculation *for*? Even where exact computation is infeasible, F is used in four ways. Practically: Chapter 4's **natural gradient** and its K-FAC implementations divide the gradient by (approximate) F, taking cautious steps along hair-trigger directions - the bowl put to work. Practically again: *elastic-weight-consolidation*-style methods use diagonal F as a per-parameter importance score, penalizing movement along high-information directions to avoid catastrophically forgetting old tasks - and the same importance reading ranks which parameters matter for a behavior. Theoretically: the Cramér–Rao bound holds whether or not anyone computes F - it is a ceiling on every probe and decoder in Part II, a statement about what the observable channel *can* reveal. And diagnostically: the spectrum of F (even sampled crudely) characterizes the local geometry of training - how many stiff directions, how many sloppy ones - which Part II's Chapter 8 uses to watch training reorganize a network's representation before its behavior shows it.

2.3 The Fisher–Rao Metric and Manifold

2.3.1 From one bowl to a landscape

Section 2.1 built one bowl at one dial setting. Now build one at *every* setting and stitch them together: declare that the squared length of an infinitesimal parameter step $d\theta$ is

$$ds^2 \;=\; d\theta^\top F(\theta)\, d\theta.$$

This makes the parameter space of the family a **Riemannian manifold**, a curved space where the local ruler varies from point to point, with the Fisher matrix as its metric tensor. Distances measured by this ruler count *accumulated distinguishability*: a path is long if the distribution changes detectably along it, short if the machine sounds the same the whole way, regardless of how far the raw dial numbers traveled. This is precisely the honest ruler the machine colony needed (Nielsen, 2018, gives a modern self-contained construction).

The payoff properties, each of which fails for the naive Euclidean ruler on dial values:

- **Reparametrization invariance.** Relabel the dial (percent vowels, log-odds of vowels, anything invertible) and all Fisher–Rao distances are unchanged. The geometry belongs to the family of distributions, not to your coordinates on it. Euclidean distance on dial values changes wildly under the same relabeling.
- **Geodesics.** The manifold has shortest paths, called **geodesics**, which bend, in dial coordinates, to route through regions where the bowls are shallow (cheap to cross) and around regions where they are steep. For the Bernoulli family the geodesic distance has a closed form,

$$d_{FR}(p,q) \;=\; 2\arccos\!\big(\sqrt{pq} + \sqrt{(1-p)(1-q)}\big),$$

which stretches near the endpoints of the dial, just where $F = 1/p(1-p)$ blows up: moving a coin from 0.98 to 0.99 is a *long* journey in distinguishability, though a short one on the dial. The program `fisher_rao_distance_bernoulli.py` plots this warping directly.

- **Uniqueness.** Chentsov's theorem (1972) singles the Fisher–Rao metric out: up to scale, it is the *only* Riemannian metric on families of distributions invariant under natural probabilistic maps (Markov morphisms - coarsenings and relabelings of outcomes). There is one honest ruler, not a menu of candidates. The invariance class is named for the same Andrei Markov as Chapter 1's property, and the link is real: Markov morphisms are built from the transition kernels his chains introduced.

2.3.2 Why a book about interpretability starts here

Two reasons. The first is training. Chapter 4's account of optimization is that training navigates this manifold: natural-gradient methods steer by the Fisher metric explicitly, and plain stochastic gradient descent turns out to respect it implicitly. None of that story can be stated without this chapter. The second is discipline. The recurring failure mode in analyzing any high-dimensional system is trusting coordinates (dial labels, neuron indices) instead of invariants. This chapter is the inoculation. Whenever this course measures an internal space of a trained network, the operating rule is the one built here: treat spaces of distributions and representations as geometric objects, and measure them with rulers that do not change when the labels do.

2.4 Contrasting Shannon and Fisher: Two Questions, Two Informations

Both quantities are called "information," and this collision of names causes confusion - but they answer different questions about different objects:

- **Shannon entropy** is a property of *one distribution*: how uncertain is the outcome? It lives on the outputs.
- **Fisher information** is a property of a *family* of distributions at a point: how sharply does the distribution change as the parameter moves? It lives on the dials or parameters inside the machine.

The Gaussian-mean example held from Section 2.2 shows they can point in opposite directions for the same object. Large σ: entropy is *high* (outcomes very unpredictable) while Fisher information about the mean, $1/\sigma^2$, is *low* (the mean is hard to estimate - the distribution barely changes shape when you slide it). Small σ: entropy low, Fisher information high. Unpredictable output and informative-about-parameters output are different properties; a distribution can spread wide over outcomes while pinning its parameters loosely, and vice versa.

	Markov/Shannon	**Fisher/Rao**
Core object	sequences, messages, one output distribution	a parametrized family of distributions
Key quantity	entropy, conditional probability, cross-entropy	Fisher matrix, Fisher–Rao distance
Question	how surprising is the outcome?	how distinguishable are nearby machines?
Lives on	outputs	parameters
Used for (this course)	the training *objective*; the n-gram ladder (Ch. 1, 3)	the training *geometry*; flat-minima analysis (Ch. 4, 8)
Zero means	outcome is certain	dial is undetectable from output

The one-line division of labor for everything that follows: *Shannon says what the model should say; Fisher says how the model should move to learn to say it.* Chapter 3 builds the machines whose outputs Shannon scores; Chapter 4 shows their tuning governed by Fisher.

Neuroscience parallel. Fisher information is a working tool of sensory neuroscience. There, the "dial" is a stimulus variable and the "machine" is a population of noisy neurons. Seung and Sompolinsky (1993) computed the Fisher information such a population carries about a stimulus, such as motion direction. The Cramér–Rao bound then gives the best decoding precision any downstream brain area can achieve. This turned a vague question (how accurately can the animal know the stimulus?) into a computable property of measured tuning curves. The same accounting applies unchanged to an artificial network. The stimulus variable is the parameter. The activation vector is the population response. Any classifier trained to read the activations is the decoder whose ceiling the bound sets. One formula prices both codes.

Takeaways

- Fisher information is the *curvature* of the Kullback–Leibler bowl around a parameter setting (distinguishability per squared unit of dial-turn) and, via Cramér–Rao, the ceiling on how precisely output can reveal its parameter.
- Stitching the bowls together yields the Fisher–Rao metric: reparametrization-invariant, geodesic-equipped, and unique by Chentsov's theorem - the one honest ruler (Riemannian metric) on a family of distributions (Nielsen, 2018).
- Shannon and Fisher measure different things (outcome uncertainty vs. parameter estimability) and can disagree on the same distribution (the wide Gaussian). The course keeps them in separate columns: objective vs. geometry.

Research Directions

From §2.2: The probe ceiling in practice

A common analysis technique is the linear probe: a classifier trained to read some concept out of a network's internal activations. The Cramér–Rao bound prices any such readout, but nobody has systematically measured how close real probes come to the ceiling. On a small model, pick a controllable input concept (sentiment of a template, language of a word), estimate the Fisher information the activations at each layer carry about the concept parameter, and compare probe accuracy against the bound, layer by layer. The gap decomposes into "information absent" vs. "readout suboptimal" - a decomposition that claims about networks constantly conflate.

From §2.3: Fisher–Rao distances between checkpoints

Practitioners compare training checkpoints by weight deltas (coordinate-dependent) or by behavioral evaluations (coarse). Compute instead a Fisher–Rao-style distance between the output distributions of successive training checkpoints, restricted to a probe corpus. Does distance-per-step spike at known events - abrupt capability transitions during pretraining, or fine-tuning boundaries? An invariant odometer for training would let different runs and different parametrizations be compared on one axis.

From §2.4: When do the two informations disagree about a checkpoint?

Take two checkpoints of one training run (or a base model and its fine-tuned sibling) and measure both informations on a fixed probe corpus: output entropy per token (Shannon) and a scalar Fisher summary via Fisher–vector products (trace or top eigenvalues). Find pairs where the orderings disagree - one checkpoint higher-entropy yet lower-information than the other. Then test which ordering predicts something downstream: robustness to prompt shift, probe accuracy, steering success. A map of where the two informations decouple would give this section's distinction operational teeth, and would tell practitioners which quantity to watch when.

Programs for Chapter 2

`kl_bowl_and_fisher.py` For a Bernoulli(p) family: numerically compute $D_{\mathrm{KL}}(p_\theta \| p_{\theta+\delta})$ for a grid of nudges δ at several base settings $\theta \in \{0.5, 0.8, 0.95\}$; overlay the quadratic $\frac{1}{2}F(\theta)\delta^2$ with the analytic $F = 1/p(1-p)$. Watch the bowl narrow as θ approaches the edge. Repeat for a Gaussian mean at several σ to see $F = 1/\sigma^2$. Two families, one law: curvature = Fisher. *Type: script + matplotlib. Deps: numpy.*

`fisher_rao_distance_bernoulli.py` Plot Euclidean distance $|p-q|$ and Fisher–Rao distance $2\arccos(\sqrt{pq} + \sqrt{(1-p)(1-q)})$ between coins, as heatmaps over (p, q) and as slices from $p = 0.5$. Mark the pairs $(0.50, 0.51)$ and $(0.98, 0.99)$: equal on the dial, far apart in distinguishability. Then verify the honest-ruler property empirically: for both pairs, plot how many coin flips a likelihood-ratio test needs to tell the machines apart at 95% confidence - the flip counts track the Fisher–Rao ruler, not the Euclidean one. *Type: script + matplotlib. Deps: numpy.*

3. Neural Networks as Probability Distribution Machines

Chapters 1 and 2 supplied a vocabulary: distributions to be shaped (Shannon's axis) and a geometry on the machines that shape them (Fisher's axis). This chapter introduces the machines themselves - and insists on a deliberately plain description of them. A generative neural network is a **probability distribution machine**: a parametrized function whose output is a probability distribution, trained to move distribution towards a target - typically in data or in instructions.

Structure of this chapter. Section 3.1 covers **sequential conditioning**: the transformer as the current highest rung of Chapter 1's ladder, replacing the counted n-gram state with a learned, content-addressable state over the full context. Section 3.2 looks at the same machine from a second angle, **attention as a directional relationship extractor**: a change of notation separates who talks to whom from what is carried, and each head becomes a relation test paired with a typed payload. Section 3.3 covers **global reshaping**: flow and diffusion models, which do not condition symbol-by-symbol at all but learn a transformation warping a simple base distribution directly into the data distribution. Section 3.4 covers **division of labor**: multi-head attention and mixture-of-experts routing, which split the shaping job across specialized sub-machines. Section 3.5 then draws the conclusion the first three sections force: the lenses of Chapters 1 and 2, powerful as they are, describe only the machine's outside - and understanding the inside requires new instruments. That closing section introduces the ideas of representations, circuits, and model biology. Throughout, the question to keep asking is not "what does the architecture look like" but "*what does it do to the distribution.*"

3.1 Transformers as Conditional Distribution Shapers

3.1.1 The ladder, resumed: from counted state to learned state

Chapter 1 ended at the state-explosion wall: conditioning on long contexts by counting fails because every long context is new. The resolution is to stop counting and start *learning*: map each context to a vector in $\mathbb{R}^d$, and make similar contexts map to similar vectors, so that evidence from one context generalizes to contexts never seen. The ladder of conditioning strategies, extended to its current top rung:

Machine	**What conditions the next-symbol distribution**	**Memory handling**
Uniform	nothing	none
n-gram (Markov)	last k symbols, by counting	hard cutoff at k
Recurrent network	one compressed hidden vector, updated each step	unbounded in principle, lossy in practice
Transformer	learned attention over *all* previous tokens	full context, content-addressable

The transformer is best read as a generalized Markov machine. It still emits one conditional distribution per position: a softmax over the vocabulary, trained by the cross-entropy objective. What changed from the previous rungs in the ladder is the state. An n-gram's state is the last k symbols, fixed in advance; a recurrent network's state is one vector that must compress everything; a transformer's state is assembled *on demand*: at each position, attention computes a weighted combination of all previous positions' representations, with weights determined by learned relevance (query–key match) rather than by recency. The state is content-addressable ("fetch whatever in the context matters for predicting the next token"), which is why transformers capture long-range dependencies that a fixed-window machine structurally cannot.

The model learns the two sides separately, per attention head: the weight matrix W_Q, building features describing what a destination position needs, and the weight matrix W_K, building features describing what a source position offers. They are kept as separate factors rather than as their product $W_Q W_K^\top$ for two reasons. The factorization is cheaper: for head width 128 inside a 4096-dimensional model, the product would be a 4096×4096 matrix while the two factors hold $2 \times 4096 \times 128$ parameters - 16× fewer - and the relation test's rank is capped at 128 either way. And the separated form is what makes the key–value cache possible at inference time: past positions' keys and values are stored once, and attention over them is not recomputed with every new token.

3.1.2 Tokens before vectors: byte-pair encoding

One step precedes every component below, and it is easy to miss because it is learned *before* training begins, not during: the text is first cut into **tokens**. Characters are too fine (the model would spend its capacity on spelling), and words are too coarse (no closed vocabulary; every rare name and misspelling falls out). The standard compromise is **byte-pair encoding** (BPE): start from characters, then repeatedly merge the most frequent adjacent pair in a training corpus until the vocabulary reaches a fixed budget - 50,257 tokens for GPT-2. Frequent strings become single tokens (*the*, *http*), rare ones split into pieces.

Three consequences of this choice return later in the book. Token boundaries are accidents of merge order, not linguistics: *token* and *token* (leading space) are different tokens, and numbers and rare strings fracture in ways that reappear as measurement artifacts in Part II's intrinsic-dimension cautions. The vocabulary size fixes the compression ratio of the embedding layer, the first map below - Part II reopens this as "the embedding is already a compression." And the training objective of Section 1.3 is charged *per token*, so cross-entropy numbers are comparable only within one tokenization - the second Research Direction of Chapter 1 makes a measurement of this.

3.1.3 The transformer decoder, component by component

The architecture used for language modeling is the multi-layer **transformer decoder** (Liu et al., 2018), a variant of the transformer (Vaswani et al., 2017), as specified for generative pretraining by Radford et al. (2018, §3.1): a multi-headed self-attention operation over the input context tokens followed by position-wise feedforward layers, producing an output distribution over target tokens. The complete computation is three equations. Given a context of k tokens $U = (u_{-k}, \ldots, u_{-1})$, written as a $k \times |V|$ matrix of one-hot rows:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \text{transformer\_block}(h_{l-1}) \quad \forall\, l \in [1, n] \\ P(u) &= \text{softmax}(h_n W_e^\top) \end{aligned}$$

where n is the number of layers, $W_e \in \mathbb{R}^{|V| \times d}$ is the token embedding matrix, and $W_p \in \mathbb{R}^{k \times d}$ is the position embedding matrix. The components, in order of application:

- **Token embedding** W_e. Row lookup mapping each vocabulary item to a vector in $\mathbb{R}^d$; the vocabulary itself is the byte-pair construction of the previous subsection.
- **Position embedding** W_p. A learned vector per position, added to the token embedding. Necessary because attention is permutation-invariant: without W_p, the model could not distinguish token order. (Vaswani et al. used fixed sinusoids; the decoder of Radford et al. learns W_p.)
- **Masked multi-headed self-attention.** Each position computes query, key, and value projections of the layer input; attention weights are the softmax of scaled query–key dot products; the output at each position is the weight-averaged values. The *mask* sets weights on future positions to zero, so position i attends only to positions $\leq i$; this is what makes the model a valid autoregressive factorization. *Multi-headed*: the d dimensions are split across n_h heads that attend independently and are concatenated and linearly projected back to $\mathbb{R}^d$; head structure is treated in Sections 3.2 and 3.4.
- **Position-wise feedforward network.** Two linear maps with a nonlinearity between (GELU in Radford et al.), inner width $4d$, applied identically and independently at every position. Attention is the only component that moves information between positions; the feedforward network transforms each position in place.
- **Residual connections and layer normalization.** Each of the two sublayers is wrapped as $x \mapsto \text{LayerNorm}(x + \text{sublayer}(x))$: the sublayer's output is added to its input, then normalized. The additive path is what makes h_l an accumulating sum of sublayer writes rather than a chain of replacements.
- **Unembedding** $W_e^\top$. The final state h_n is projected back to vocabulary logits by the transpose of the input embedding (weights tied), and a softmax yields the next-token distribution. This is the q whose cross-entropy against data is the training objective of Section 1.3; equation (1) of Radford et al. is that objective, summed over positions.

Reference hyperparameters, for scale (Radford et al., 2018): $n = 12$ layers, $d = 768$, $n_h = 12$ heads, feedforward inner width 3072, context $k = 512$, byte-pair vocabulary of 40,000.

3.1.4 The residual stream: the transformer's shared working memory

The **residual stream** is a running d-dimensional vector at each position - the h_l sequence - which every sublayer reads from and adds into; it is the shared channel through which intermediate results travel. The **attention head** is the smallest separable unit of the attention operation. Whatever the trained model computes, it computes by heads and feedforward layers writing into and reading from the residual stream; understanding a trained transformer means understanding what travels through that stream - a task taken up in Section 3.5. This one vector is the object Part II measures, probes, decomposes, and steers; nearly every method there is an operation on the residual stream.

3.1.5 From transformer to Generative Pretrained Transformer (GPT)

In 2018 the standard approach in language processing was one model per task. Entailment, question answering, similarity, classification: each had its own labeled dataset, and often its own architecture. Labeled data is scarce and expensive. Unlabeled text is nearly unlimited. The problem: how can a model learn from unlabeled text in a way that carries over to tasks with few labels?

The GPT paper gave a two-stage answer. Stage one: train a language model on a large corpus of plain text. The objective is next-token prediction, which needs no labels. Stage two: for each task, reuse that model. The task input is rewritten as a token sequence, the sequence is fed through the trained network, and a single linear layer converts the final state into the task answer. A short fine-tuning run adapts the weights.

Each word of the name records one decision. *Generative*: the pretraining objective is generation, chosen because prediction of the next token is the one task the raw text itself supervises. *Pretrained*: the model is trained before any task is known; the tasks arrive later and share the same weights. *Transformer*: the network chosen for the job. The original transformer was built for translation and had two halves, an encoder for the source sentence and a decoder for the target. Generation has no source sentence, so GPT kept only the decoder (Section 3.1.3).

The result: one pretrained model, with one linear layer added per task, improved the state of the art on 9 of 12 benchmarks.

The result matters for this book because of what it implies about the model's internal vectors. A linear layer cannot compute entailment on its own. It can only read out information that is already present, and present in linear form, in the states the pretraining built. Two ablations in the paper support this reading. Transferring more pretrained layers helps monotonically, so every layer holds usable structure, not just the last. And performance on downstream tasks rises steadily during pretraining, before any task data is seen, so the representations improve on their own. Next-token prediction, pushed far enough, manufactures representations that other tasks can use. (Radford, Narasimhan, Salimans, and Sutskever, 2018; Vaswani et al., 2017.)

3.2 Attention as a Directional Relationship Extractor

3.2.1 Two jobs, cleanly separated: routing and payload

Section 3.1 described attention operationally: queries, keys, values, a softmax, a weighted average. A change of notation reveals more structure than the operational description shows. The textbook computation runs in three steps: compute a value vector per position, mix the values across positions using the attention pattern, and project the mixture back into the stream. Written on the whole sequence at once, with X the matrix of residual vectors (one row per position), the routing is computed first:

$$A \;=\; \mathrm{softmax}\Big(X W_Q W_K^\top X^\top\Big)\,. \tag{3.1}$$

A is the **attention matrix**: one row per destination position, one column per source position, each row a distribution over sources. The head's output is then

$$h(X) \;=\; (A \otimes W_{OV}) \cdot X, \qquad W_{OV} = W_O W_V. \tag{3.2}$$

A mixes across tokens, while W_{OV} acts on each token's vector independently. The tensor product $\otimes$ states exactly that the two factors touch different axes of X: A the position axis, W_{OV} the feature axis. Two matrices, two axes, two complete jobs. A selects, for every destination position, the source positions and their weights. That is the routing, complete in itself. W_{OV} selects what is read out of a source's residual vector and how it lands in the destination's. That is the cargo handling, complete in itself. The observation, in the words of the paper that introduced this rewriting: "which tokens to move information from is completely separable from what information is 'read' to be moved and how it is 'written' to the destination" (Elhage et al., 2021). Because the two factors act on different axes, each can be studied on its own.

3.2.2 The QK side is a directed relation test

The score between destination i and source j is a bilinear form,

$$s_{ij} \;=\; x_i\, W_Q W_K^\top\, x_j^\top.$$

Three properties of this object justify calling the head a relationship extractor, and a directional one.

It is a test on *pairs*. The score takes two arguments, a destination and a source, and measures their fit under the head's relation; every routing decision the head makes comes from these pairwise fits.

It is *directed*. $W_Q W_K^\top$ is a general square matrix, free to differ from its transpose, and trained heads use the freedom. Section 3.1 already named the two sides: W_Q builds features describing what the destination needs, W_K features describing what a source offers. So s_{ij} and s_{ji} are separate tests: subject-of and has-subject are different relations, and the machinery represents each on its own terms.

It is *soft*. The softmax turns each destination's row of scores into a distribution over sources: a weighted, uncertainty-carrying instance of the head's relation, with the weights as its confidence profile.

3.2.3 The OV side is the relation's typed payload

W_{OV} decides what a detected edge carries. Its singular value decomposition says how much: the head bottleneck gives $\text{rank}(W_{OV}) \leq d_{\text{head}}$, so the head reads at most a d_{head}-dimensional subspace of the source's residual vector and writes at most a d_{head}-dimensional subspace of the destination's. A head is a typed channel: this relation, when detected, moves this kind of content and deposits it there. Copying heads are the identity-like special case, with a large positive spectrum on a token-identity subspace; heads with large negative W_{OV} eigenvalues are the erasing case, writing the negative of what they read (Elhage et al., 2021). A relation family's payload has at most d_{head} dimensions of vocabulary, the same rank bound Section 3.1 noted for the $QK^\top$ product.

3.2.4 The layer builds a graph

The two sides recombine into a compact picture of a whole attention layer. For each head, build a directed weighted graph over positions: the head's A, constructed fresh for each input. Along every edge, transport the head's typed payload: apply W_{OV} to the source vector, scale by the edge weight, add into the destination. Then sum over heads, which act independently and additively. A transformer layer is dynamic graph construction followed by linear message passing, and the feedforward network that follows is per-position computation on the aggregated messages. Each clause of this picture is one factor of the rewritten equation: the picture is the algebra, restated. It also says what training a head means: choosing one bilinear relation test and one typed channel, jointly.

One corollary reshapes how the residual stream should be read. Heads copy subspace-loads of content between positions, so the residual vector at a position is a composite: alongside content about its own token, it carries linear subspaces imported from other positions. The phrase "contextual word embedding" describes one part of this composite; the imported subspaces are the rest.

Depth composes the extractors. Later heads build their queries and keys from earlier heads' outputs, so a relation test can take, as its arguments, the outputs of an earlier relation. Part II's circuits chapter dissects the canonical case: an early head extracts the previous-token relation and stamps each position with its predecessor's identity; a later head matches the current token against those stamps, extracting a relation between relations.

Three boundaries keep the framing calibrated. Heads and relations map many-to-many: one bilinear form can serve several relations in different regions of representation space, and several heads can share one relation. A records who communicated with whom and how strongly; the meaning of the communication lives in the payload and in what downstream components do with it. And the softmax builds each destination's edges under a fixed budget: the weights in a row sum to one, so sources compete, and a source's mass reflects the competition as much as its own fit.

3.3 Flow and Diffusion Models as Global Reshaping

3.3.1 Shaping without conditioning

Sequential conditioning is not the only way to place probability mass on structured outputs. The second strategy abandons the symbol-by-symbol factorization entirely: start from a distribution

that is trivial to sample (a standard Gaussian) and learn a transformation T that warps it into the data distribution. Sampling is then one draw of noise pushed through T. Where the transformer shapes the distribution *one conditional at a time*, a flow model shapes it *all at once*, as a global change of variables:

$$p_{\text{model}}(x) \;=\; p_{\text{base}}\big(T^{-1}(x)\big)\left|\det \frac{\partial T^{-1}}{\partial x}\right|,$$

the Jacobian factor accounting for how the warp stretches and compresses probability mass - geometry doing the bookkeeping that conditional probabilities did in Chapter 1.

Two families realize the idea at scale. **Diffusion models** (Ho, Jain, and Abbeel, 2020) learn the warp implicitly: corrupt data by gradually adding Gaussian noise until nothing remains, train a network to undo one small step of corruption, and generate by iterating the denoiser from pure noise: a walk from the base distribution to the data distribution in hundreds of small steps. **Rectified flows** (Liu, Gong, and Liu, 2022) learn it explicitly as a velocity field: pair noise samples with data samples, draw straight lines between them, and train a network to follow those lines, so that generation is numerical integration of an ordinary differential equation - ideally in very few steps, since straight paths are cheap to integrate.

3.3.2 The contrast that matters: conditioning versus warping

	Sequential (transformer)	**Global (flow/diffusion)**
Shaping mechanism	one learned conditional per symbol	one learned warp of the whole density
Where probability lives	explicit: softmax each step	implicit: base density × Jacobian
Sampling	token by token, left to right	noise in, sample out (possibly many refinement steps)
Natural domain	discrete sequences (text, code)	continuous spaces (images, audio, latents)
Chapter-1 ancestor	the n-gram ladder	none - a genuinely different lineage
Fisher-geometry connection	training objective is cross-entropy on conditionals	the warp itself is mass transport across the space

The point of including flows in this course: they demonstrate that "shaping a distribution" is the invariant task, with conditioning merely one strategy for it. A transformer sculpts the distribution through a factorization; a flow sculpts it through a map. Both end at the same place - a machine whose dials parametrize a distribution, whose quality Shannon's cross-entropy scores, and whose dial-space carries the Fisher geometry of Chapter 2. Training either machine is motion across that geometry, which is where Chapter 4 picks up.

3.4 Mixture-of-Experts and Multi-Head Structure

3.4.1 Dividing the shaping job

The third strategy is orthogonal to the first two: instead of changing *how* the distribution is shaped, divide *who* shapes it. Modern architectures split the work along two axes.

Multi-head attention (Vaswani et al., 2017) runs many small attention operations in parallel within each layer, each with its own learned query, key, and value projections - each head free to attend by different criteria (recency, syntax, coreference, matching brackets) and to read and write its own low-dimensional slice of the residual stream. The full layer's contribution is the sum of its heads: the conditional distribution is shaped by a *committee*, each member specializing in one kind of relevance.

Mixture-of-experts routing (Shazeer et al., 2017) applies the same idea to the feed-forward layers, at coarser grain and with a twist: of N expert sub-networks per layer, a learned gate activates only the top-k (often 2) per token. Parameters scale with N; compute scales with k. The gate makes computation *conditional*: different tokens literally flow through different sub-machines, so the distribution-shaping function is now a routed composition rather than a fixed pipeline.

3.4.2 Why the division of labor matters for understanding

The committee structure is not just an efficiency trick. It determines what units of analysis exist when you try to understand the trained machine:

- *Heads are separable.* One can silence a single head, or read its attention pattern, and ask what that one committee member does. Sometimes there is a clean answer: heads have been found that track a token's predecessor, match closing brackets to their openers, or copy repeated patterns forward.
- *The residual stream is shared.* Many heads and layers write into the same d-dimensional workspace. What the model "knows" at any moment is therefore spread across the sum of many contributions, and no single neuron need correspond to any single concept.
- *Expert routing is discrete.* In a mixture-of-experts model, different tokens flow through different sub-networks. Any account of what the model does is conditional on the routing path taken.

The design lesson runs both directions. Architecture determines what kinds of explanation are even possible. And empirical findings about which specializations actually emerge reveal how the architecture's degrees of freedom are actually used.

Neuroscience parallel. Division of labor among specialized sub-machines is an organizational fact known about the cortex. Distinct brain regions handle faces, places, motion, and language. Lesion evidence established this in the 19th century. Functional imaging later mapped it noninvasively, using contrast designs: show faces versus objects, and the region that cares about the difference reveals itself (Kanwisher et al., 1997, for the fusiform face area). The corresponding objects in the artificial network are attention heads and routed experts: sub-networks recruited conditionally by input type. A shared caution transfers with the parallel. In both substrates,

specialization is a matter of degree, and clean one-region-one-function stories usually dissolve on close inspection.

3.5 The Case for Interpretability

3.5.1 Two powerful lenses, one shared blind spot

Part I has equipped us with two lenses, and it is worth being precise about what each one sees. The Markov/Shannon lens describes the machine's *output*: a conditional distribution over the next symbol, scored by cross-entropy, improved by richer conditioning. The Fisher/Rao lens describes the machine's *dials*: how sensitively the output distribution responds to parameter changes, and what a principled distance in dial-space looks like.

Both lenses share a blind spot. Each treats the network as a single, undivided probability-emitting function. Shannon's quantities are computed from the output distribution alone; they say nothing about how attention heads, feed-forward layers, or experts produced that distribution. Fisher's geometry lives on the parameters as one undifferentiated vector; after training it does not decompose into statements like "this head, in this layer, handles subject–verb agreement."

Neither lens tells you where the fact is stored, which components retrieved it, what else is entangled with it, or how to change it without retraining the whole machine as their resolution is too coarse. New instruments are required for answering these questions.

3.5.2 Opening the machine: representations and circuits

The new instruments start from the two anatomical facts established in Sections 3.1 and 3.3, and turn each into an object of study.

Representations. At every token position, at every layer, there is an activation vector in the residual stream. These vectors are the model's working notes. A central empirical finding is that these notes have usable structure - in particular, many human-recognizable concepts (a language, a sentiment, a topic, the truth of a statement) correspond approximately to *directions* in activation space. This makes reading the notes possible by projecting the activation onto a direction and obtaining a measurement of a concept.

Circuits. The committee members of Section 3.4 (heads, layers, experts) connect through the residual stream into small functional assemblies: one head marks where a pattern occurred earlier, another copies what followed it, and together they implement "repeat what came after the last occurrence." Such an assembly is called a *circuit*: a subgraph of components implementing an identifiable algorithm. Circuits are the natural candidate for the level of description both lenses lack - coarser than a parameter, finer than the whole machine, and stated in terms of what is computed rather than merely what is output.

Interventions. Unlike any physical system of comparable interest, an artificial network is fully observable and fully writable: every activation can be recorded, every component silenced, every direction added or removed, mid-computation, at will. Hypotheses about representations and circuits can therefore be tested *causally* rather than argued from correlation: silence the candidate component and watch the behavior; edit the candidate direction and watch the concept.

3.5.3 Model biology

A trained network is not designed; it is *grown*. Billions of parameters are set by an optimization process. The right stance toward such an object is the biologist's stance toward an organism: observe behaviors, form competing hypotheses, intervene, and let the interventions decide. Neuroscience spent a century building this toolkit for brains (receptive-field mapping, lesion studies, population-level analysis), and some of it transfers along lessons about where each method misleads.

The questions are: what do the activation vectors represent, and how are they arranged? Which assemblies of components implement which behaviors? How can behavior be changed by editing the inside rather than retraining the whole? And when an explanation is offered, how do we validate it is true of the machine rather than merely plausible. These four questions (representation, circuits, control, and evidence) define the second half of this course.

Takeaways

- A generative network is a distribution machine; architectures differ in shaping strategy. The transformer is the learned successor of the n-gram ladder: a content-addressable Markov state over the full context, emitting one softmax conditional per position (Vaswani et al., 2017).
- Each attention head factors into two independent parts: a directed relation test on pairs of positions (A, from $W_Q W_K^\top$) and a typed payload of rank at most d_{head} (W_{OV}). A layer builds a directed graph per input and passes linear messages along it; the residual vector at a position is a composite of its own content and imported subspaces (Elhage et al., 2021).
- Flow and diffusion models shape globally instead: a learned warp from base Gaussian to data distribution (Ho et al., 2020; Liu et al., 2022) - proof that conditioning is one strategy for the invariant task, not the task itself.
- Multi-head attention and mixture-of-experts routing divide the shaping job across specialized sub-machines (Shazeer et al., 2017), creating the units (heads, experts, the shared residual stream) that any account of the machine's inner workings must be written in.
- The Markov/Shannon and Fisher/Rao lenses see only the machine's outside: output distribution and parameter sensitivity. Understanding the inside requires new objects (representations, circuits, causal interventions) studied with a biologist's stance toward a grown, undocumented system.

Research Directions

From §3.1: Where does the learned state beat the counted one?

The transformer's advantage over n-grams must be localized in *which tokens* it predicts better. Compare a small transformer against order-2 through order-5 word models token-by-token on shared text; cluster the tokens where the transformer's advantage is largest. Do the clusters correspond to identifiable long-range computations - agreement across clauses, coreference, bracket matching? This builds a null model that any explanation of the machine needs: a mechanism claim is only interesting for behavior the counted state cannot already explain.

From §3.3: One geometry for two shaping strategies

Transformers and flows shape distributions by different means; do they arrive at similar internal structure? Train both a small autoregressive model and a small flow or diffusion model on the same low-dimensional distribution, then compare their internal representations, using dimensionality profiles of the activations and representational similarity analysis. Convergent structure across shaping strategies would suggest the data, not the architecture, dictates what forms inside - a foundational question for how far any analysis of one model family transfers to another.

From §3.4: Does routing buy cleaner components?

The cleanest architectural question raised by this chapter: for matched capability, do mixture-of-experts models pack fewer unrelated features into each neuron than dense models - i.e., does the router absorb specialization that a dense model must squeeze into shared dimensions? Train small dense and mixture-of-experts models on the same corpus and compare, per component, how many distinct input types drive each unit. Either answer matters: cleaner experts would make routing an implicit aid to understanding; equally mixed experts would show the packing pressure comes from something other than raw dimension scarcity.

Programs for Chapter 3

`ngram_vs_tiny_transformer.py` Train a 2-layer, 2-head character transformer (∼0.5M parameters, torch, one laptop-hour on CPU) on the Chapter-1 corpus, alongside order-2 through order-5 character Markov models. Report held-out per-character cross-entropy for all machines on one bar chart - the Chapter-1 entropy ladder extended by one learned rung. Then print the ten test positions where transformer and best n-gram disagree most, and look at them: they are almost always long-range dependencies. *Type: training script + matplotlib. Deps: torch, numpy.*

`flow_warp_2d.py` Implement a minimal 2-D normalizing flow (four affine coupling layers, ∼100 lines of torch) and train it to map a standard Gaussian to a two-moons distribution. Animate the warp: plot the image of a regular grid and of 1,000 base samples after each coupling layer, watching the Gaussian bend into two moons. Overlay the exact log-density from the change-of-variables formula. Global reshaping, watched frame by frame. *Type: training script + matplotlib animation. Deps: torch, numpy.*

`expert_specialization_toy.py` Build a mixture-of-experts regressor: four small multilayer perceptrons and a learned soft gate, trained on data drawn from four visibly different regimes (four quadrants of the plane with different functions). Plot (a) the gate's routing probabilities over the plane and (b) each expert's error restricted to each regime, across training. Watch specialization emerge from a uniform start - then break it: repeat with top-1 hard routing and observe expert collapse (one expert takes everything), the failure mode that load-balancing losses exist to prevent. *Type: training script + matplotlib. Deps: torch, numpy.*

4. Training Dynamics

Chapter 2 built a geometry on the dials of a probability machine. Chapter 3 built the machines. This chapter puts the two together. The claim: training is motion across the Fisher geometry. Three pieces of evidence support it. One method uses the geometry deliberately (natural gradient descent). One engineering idea makes the geometry affordable (Kronecker factorization). And plain stochastic gradient descent, which computes none of it, still behaves as if it knew the geometry. That last fact is the chapter's central puzzle.

Structure of this chapter. Section 4.1 derives **natural gradient descent**: "steepest descent" is undefined until a metric is chosen, and the Fisher metric gives a specific update rule. Section 4.2 covers **cost**: the full Fisher matrix of a large model cannot be stored or inverted, and the approximations that made curvature usable form a ladder, with Adam at the bottom and K-FAC higher up. Section 4.3 turns to the puzzle: **stochastic gradient descent** computes no curvature yet finds flat minima, and flatness, measured correctly, is a Fisher quantity. Section 4.4 asks a further question: if training builds the structure inside the machine, can training be chosen so the structure comes out legible and controllable? Section 4.5 traces how the architecture itself keeps the training signal alive across depth, what second instability the fix must trade against, and what the fix created. Claims in Sections 4.1–4.2 are mathematics and engineering. Claims in Sections 4.3–4.4 are partly active research and are flagged as such.

4.1 Natural Gradient Descent

4.1.1 Steepest descent is metric-dependent

Gradient descent updates parameters by $\Delta\theta = -\eta\nabla_\theta L$: a step against the gradient. The usual justification is that the gradient is the direction of steepest ascent. The justification is incomplete. Steepest per unit of what? The gradient is steepest per unit of *Euclidean* distance in parameter space, and Chapter 2 showed that this ruler is not invariant. A fixed-size parameter step changes the output distribution by different amounts depending on where you stand and on how the dials are labeled. So plain gradient descent takes its largest steps where the distribution is most sensitive and its smallest steps where it is least sensitive, which is backwards. Its trajectory even changes if the parameters are merely relabeled.

The repaired question: which direction gives the most loss reduction per unit of *distribution change* - per unit of Fisher–Rao distance, the invariant ruler of Section 2.3? The answer has a closed form, the **natural gradient**:

$$\Delta\theta \;=\; -\eta\, F(\theta)^{-1}\nabla_\theta L.$$

The inverse Fisher matrix rescales each direction by its own sensitivity. Loose dials get large steps. Sensitive dials get small ones. Correlated dials are decorrelated. The update is invariant to reparametrization: percent-vowels coordinates and log-odds coordinates produce the same trajectory through distribution space. Plain gradient descent has no such guarantee. (Amari, 1998; Martens, 2014, covers the modern theory, including the connection to Gauss–Newton methods and the sense in which natural gradient behaves like a second-order method.)

4.1.2 A two-dial example

An example small enough to compute by hand. Fit a Gaussian to data by adjusting (μ, σ). The Fisher matrix is diagonal with entries $1/\sigma^2$ and $2/\sigma^2$. Both entries grow as σ shrinks: in a narrow distribution, a fixed nudge of the mean is large relative to σ, and the old and new distributions barely overlap. Plain gradient descent does not see this. As σ shrinks, its effective step size in distribution space grows as $1/\sigma^2$, and it oscillates. The learning rate must then be set small enough for the worst region, which slows progress everywhere else. Natural gradient multiplies by $F^{-1} = \mathrm{diag}(\sigma^2, \sigma^2/2)$ and moves at the same speed at every scale. The program `natural_vs_plain_gradient.py` draws both trajectories. One path zigzags into the narrow-σ region and stalls. The other follows, approximately, a Fisher–Rao geodesic.

4.2 Making It Tractable: Kronecker-Factored Approximate Curvature

4.2.1 The impossible Fisher matrix

For a model with n parameters, F is an $n \times n$ matrix. At $n = 10^9$, storing it takes 10^{18} numbers, and inverting it takes about 10^{27} operations. Neither is possible. So every practical use of Fisher geometry rests on approximating F^{-1} cheaply, and the approximations form a ladder.

- **Diagonal.** Keep only the diagonal of F: one sensitivity per parameter, all correlations dropped. This is where everyday optimizers live. Adam rescales each parameter's step by a running average of its squared gradients, which is, in expectation, a diagonal Fisher preconditioner. Anyone who trains with Adam is running a crude natural gradient (Kingma & Ba, 2015).
- **Kronecker-factored (K-FAC).** One level up, keep the correlations within each layer. A layer computes Wa: a weight matrix times an incoming activation vector. The gradient of the loss with respect to W is therefore an outer product of two vectors - the incoming activations a and the gradient g arriving from the layers above. Because every sample's gradient has this shape, the layer's block of the Fisher matrix is, to a good approximation, a Kronecker product of two small matrices: $F_{\text{layer}} \approx A \otimes G$, where A is the covariance of the activations and G is the covariance of the arriving gradients. The payoff is inversion: $(A \otimes G)^{-1} = A^{-1} \otimes G^{-1}$. For a 1000×1000 weight matrix, the exact Fisher block is $10^6 \times 10^6$ and cannot be inverted; the two factors are 1000×1000 each and can. The result is an approximation of the Fisher that is neither diagonal nor low-rank, costs only a few times more per step than the plain gradient, and makes enough extra progress per step to beat stochastic gradient descent with momentum in practice (Martens & Grosse, 2015).
- **Full.** Possible only for the toy models in this book's programs. That is what makes them instructive.

4.2.2 What curvature-aware training is for

Where does this matter in practice? Large-scale pretraining is dominated by first-order methods: Adam, normalization layers, tuned learning-rate schedules. Curvature-aware methods survive where per-step quality matters more than per-step cost. Reinforcement learning from human feedback is one such place; its trust-region ancestry is Fisher preconditioning. Fine-tuning under tight step budgets is another. Badly conditioned problems are a third. For this course, K-FAC matters conceptually as much as practically. It shows that the Fisher geometry of Chapter 2 is not philosophy. It is an engineering resource: pay for curvature, and convergence measurably improves. That sharpens the puzzle of the next section. Why does the method that pays nothing still inherit the benefits?

A newer optimizer shows the same thinking in a cheaper form. The Muon optimizer (Momentum Orthogonalized by Newton–Schulz) accumulates the momentum-averaged gradient of the loss with respect to each weight *matrix* and then orthogonalizes (in fact orthonormalizes) the update, using Newton–Schulz iterations, before applying it. Orthogonalization sets every singular value of the update to one: $U\Sigma V^\top$ becomes $UV^\top$. The result is that dominant singular values are damped, weak ones amplified, and the step spends equal effort in every singular-vector direction of the matrix. It ignores the relative sizes of the singular values and treats their directions as equal. This contrasts with K-FAC which estimates the Fisher matrix and inverts it; Muon never computes any curvature. But both act on the same diagnosis, which is that in a space of a billion dimensions, the raw gradient is a poor guide, because a step of fixed size means different amounts of behavioral change in different directions. Information geometry names the correction - measure a step by its effect on the output distribution, not by its length in parameter coordinates. K-FAC applies the correction through an explicit metric; Muon applies a rough version of it through matrix normalization, at almost no overhead. The rough version scales: Muon has trained language models at pretraining scale at roughly twice the computational efficiency of a tuned Adam baseline (Jordan et al., 2024; Liu et al., 2025). Why this should work is not well understood; one possibility is that in large spaces the singular values can be trusted less than the singular-vector directions. A reading is that because of high condition numbers, useful directions for loss reduction are ignored merely because of their lower relative singular values, and that this is a mistake - those directions are still important. Another finding is that it works better when applied to the Q, K, and V matrices individually than to their products (Boza). A good discussion of theoretical and experimental analyses is at https://kellerjordan.github.io/posts/muon/.

A recent paper gave a measured take on Muon. If you keep the update's singular directions and replace its singular values with random noise, the performance at GPT-2 scale barely changes: flattened, randomized, and inverted spectra all train comparably well (Shumaylov et al., 2026). The singular values evidently carry little signal. But what the normalization in Muon buys is a stable step size: one learning rate stays near optimal for the whole run, where plain gradient descent would need a varying schedule.

4.3 Stochastic Gradient Descent's Implicit Geometric Bias

4.3.1 The flat-minima observation

A deep network's loss surface has many minima. Their training losses are nearly identical. Their test performances are not. The empirical pattern: small-batch stochastic gradient descent finds

flat minima, where the loss rises gently under parameter perturbation, and flat minima generalize better. Large-batch training finds sharp minima, which generalize worse. The intuition is that a wide basin tolerates the difference between training and test distributions, the way a machine sitting in a broad bowl of Chapter 2 keeps producing nearly the same output under parameter jitter. (Keskar et al., 2016, documented the pattern in the large-batch setting.)

4.3.2 The objection that makes it geometric

There is a hole in the naive version of this story. Flatness measured by the Hessian in raw parameter coordinates is not reparametrization-invariant. Rescale one layer's weights up and the next layer's down. The network computes the same function, so its generalization is unchanged - yet the same minimum can now be made to look arbitrarily sharp or arbitrarily flat. Parameter-space flatness is an artifact of coordinates. (Dinh et al., 2017.)

Chapter 2 said what to do with coordinate artifacts: replace the Euclidean ruler with the Fisher ruler. The repaired quantity is Fisher-weighted sharpness, schematically $\mathrm{tr}\big(F(\theta^*)^{-1}\nabla^2 L(\theta^*)\big)$. It measures the loss's curvature per unit of distribution change, not per unit of dial-turn. It is invariant to relabeling. And it correlates with generalization where the raw Hessian can be gamed. The flat-minima story survives, but only in Fisher coordinates: what matters is flatness on the manifold of distributions, not in parameter space. (Status: the invariance repair is settled mathematics. Which invariant sharpness best predicts generalization is active research.)

4.3.3 Why plain stochastic gradient descent finds them

One question remains: mechanism. Stochastic gradient descent computes no Fisher matrix. Why would it prefer Fisher-flat minima? The leading account is the gradient noise. Mini-batch gradients are noisy, and the noise is not uniform across directions: its covariance is approximately the empirical Fisher, largest exactly where the output distribution is most sensitive. Training then behaves like a ball rolling on the loss surface while being shaken, and the shaking is hardest along the steepest walls. Narrow bowls eject the ball. Wide bowls keep it. Settling in Fisher-flat basins is not a preference the optimizer computes; it is the steady state of its noise. The same geometry the optimizer never touches acts on it as an implicit regularizer. (Status: supported by analyses of the noise covariance and by controlled experiments; a complete quantitative theory for realistic architectures is open.)

Neuroscience parallel. Training noise has a biological counterpart, and the counterpart is deliberate. Juvenile songbirds learn to sing by practicing thousands of times a day. They never repeat the song the same way twice; every rendition varies. That variability is not motor sloppiness. It is injected on purpose, by a dedicated forebrain nucleus called LMAN. Ölveczky, Andalman, and Fee (2005) proved this by direct intervention: silence LMAN, and the juvenile's song immediately becomes stereotyped. The bird stops exploring, and song learning stalls. The point is simple and strong - the variability *is* the learning mechanism, not noise on top of it. The corresponding object in network training is mini-batch gradient noise. In both systems, structured randomness drives the search across the space of solutions. And in both, removing it (large-batch training in one case, LMAN inactivation in the other) produces convergence that looks cleaner and learns worse.

4.4 Training-Time Levers on Interpretability

4.4.1 Structured is not legible

Sections 4.1–4.3 showed that training fills the machine with structure. The structure is not arranged for a reader. When data offers more useful features than the network has dimensions, the loss-optimal solution packs several features into shared dimensions, so a single neuron fires for a mixture of unrelated meanings (*Toy Models of Superposition*; Elhage et al., 2022). The same compression that creates the structure hides it.

A compiled binary stripped of debugging symbols is a close analogy, on both counts. Stripping removes the names. Optimization packs the substrate: compilers inline, deduplicate, and reuse registers, so one machine address serves several source-level meanings, as one neuron serves several features. Reverse engineering stays possible because the structure survives; it is hard because the names are gone and the substrate is shared.

The way forward turns the problem around. Legibility is a property of the trained solution, and the trained solution is selected by training choices, so training choices set legibility. The choices form a space, and the research divides into three regimes of that space, defined by two parameters: whether legibility is a goal of the choices at all, and how specific its target is.

The binary analogy extends to all three regimes, and it exposes the trend. Software engineering did not stop at disassembling shipped binaries. It learned to compile with debug symbols, and then to design programs for observability: asserts, logging, telemetry at chosen points. Interpretability is retracing that path. Part II analyzed finished artifacts. The work of this section moves earlier: first noticing that build settings already affect readability, then compiling for readability, then instrumenting chosen capabilities from the start. The instruments are migrating upstream, from after training to inside it.

- **Regime 1: incidental.** Legibility is not optimized. It varies anyway, as a side effect of hyperparameters chosen for other reasons. The dials: weight decay (whether a readable algorithm or an opaque lookup table survives training), dropout (whether redundant backup circuits form), width and data sparsity (how much feature-packing occurs). The claim: interpretability outcomes are already being set, silently, by choices nobody audits. The analogy: compiler flags chosen for speed, deciding readability as a side effect.
- **Regime 2: deliberate, whole-network.** Legibility of the entire model becomes an explicit training objective. The dials are architectural: activation functions that favor one-meaning-per-neuron, discrete bottlenecks whose codes are enumerable, architectures whose decomposition is fixed before training. The claim: legibility can be bought at training time, at some capability cost, and the purchase must be audited, because computation can reorganize to satisfy the metric while staying opaque. The analogy: compiling with debug symbols.
- **Regime 3: deliberate, targeted.** The target narrows from every neuron to designated concepts and capabilities: place them in chosen locations, keep them linearly encoded, make them removable. The dials operate inside the training loop: masked gradients that confine a capability to a region, projections that delete concept directions during fine-tuning, trait directions added so the model never encodes them. The claim: localization and encoding need not be discovered after training; they can be specified before it. The analogy: designing the program for observability.

Reading down the list, intent rises and scope narrows: no target, then the whole network, then designated concepts. The strength of evidence falls in the same order. Regime 1's effects are demonstrated in small models. Regime 2 has working methods and one well-documented failure mode. Regime 3 is the youngest and the most active. The next three subsections give the evidence.

4.4.2 Regime 1: incidental - hyperparameters silently set the representation

Three hyperparameters, each chosen for reasons unrelated to interpretability, each documented to move it.

Weight decay can select the legible algorithm. On small algorithmic tasks, a network first memorizes its training set. Much later, and abruptly, it starts generalizing. The phenomenon is called grokking. Nanda et al. (2023) reverse-engineered it on modular addition. During the long plateau, the network is quietly assembling a general algorithm, built from Fourier components and trigonometric identities, while the memorized solution still drives behavior. Weight decay then strips the memorization away, and the general algorithm takes over. Remove weight decay and the transition largely fails to happen. A regularization constant decided whether the trained network contained a readable algorithm or an opaque lookup table.

Dropout breeds redundancy. Dropout trains every component to survive its neighbors' random failure. The plausible product is networks full of backup components - circuits that lie dormant until a primary pathway is damaged, then take over its function. Redundancy of this kind is directly hostile to causal analysis: silence a component, observe no change, and wrongly conclude it did nothing.

Width and data sparsity set the packing pressure. In the toy models of superposition cited above, how much feature-packing occurs is controlled by the ratio of useful features to available dimensions and by how sparsely features occur - quantities set by architecture and data curation. Legibility per neuron is, in part, a capacity-planning outcome.

The list is surely incomplete. Nobody has checked, for instance, whether models trained with different optimizers are equally analyzable. That measurement problem appears in the research directions at the end of this chapter.

4.4.3 Regime 2: deliberate - training for legibility

Four methods define regime 2's record: real gains, a recurring cost, and one instructive trap.

- **Changed activation functions.** Elhage et al. (2022, "Softmax Linear Units") replaced the standard MLP nonlinearity with one designed to encourage basis-aligned, one-meaning-per-neuron representations. It worked (the fraction of interpretable neurons rose), but with one important complication. Some features did not become interpretable; they *relocated*, hiding in a normalization step where the interpretability metric was not looking. This is the field's cautionary tale: optimize a proxy for legibility, and the computation may reorganize to satisfy the proxy while staying opaque.
- **Discrete bottlenecks.** Tamkin et al. (2023) train networks whose hidden states must pass through a codebook: each token's state becomes a small set of discrete codes. The codes are sparse, enumerable, and individually inspectable, and flipping them changes behavior predictably - legibility and a control surface, bought at a modest capability cost.

- **Interpretable-by-architecture models.** Backpack language models (Hewitt et al., 2023) constrain predictions to be non-negative weighted sums of non-contextual word-sense vectors. The decomposition is not discovered after training; it is the architecture. Editing a sense vector produces predictable global changes in behavior.
- **Engineering away the packing.** Jermyn et al. (2022) show, in miniature models, that training adjustments (initialization and bias choices) can push otherwise-identical networks into one-meaning-per-neuron solutions at little or no loss penalty - evidence that some of the packing of regime 1 is contingent on training details rather than forced by the task.

The economics are consistent across these methods. Legibility can be bought at training time. The price is some capability - the "interpretability tax." And the SoLU episode shows the purchase must be audited: a metric improved is not a machine understood. As of this writing, no frontier production model is trained primarily for legibility, because the tax competes directly with capability scaling. Whether the tax is fundamental or an artifact of immature methods is open.

4.4.4 Regime 3: deliberate - training for steerability

Regime 3's claim was the strongest: what training encodes, and where, can be specified in advance. Before the methods, the question of why that should be possible at all. Behavior can be steered by adding vectors to a network's internal activations because many high-level concepts are encoded as linear directions. There is theoretical work arguing that this linearity is itself a product of training: in simplified settings, the implicit bias of gradient descent plus the log-odds structure of the next-token objective provably yields linear encodings (Jiang et al., 2024). If that account is right, steerability is not an architectural accident. It is a product of the training dynamics of Sections 4.1–4.3, and what training produces, training can be made to produce on purpose.

The recent research does so from three directions:

- **Placing capabilities.** Gradient routing (Cloud et al., 2024) masks gradients during training so that designated data (say, a capability you may later want to remove) updates only a designated subregion of the network. The capability is localized by construction; deleting or gating that subregion later removes it cleanly. Localization stops being something you hope to discover and becomes something you specify.
- **Steering generalization itself.** Fine-tuning on narrowly flawed data (insecure code) can produce broadly misaligned behavior, a phenomenon called *emergent misalignment* (Betley et al., 2025). Concept-ablation fine-tuning (Casademunt et al., 2025) applies an intervention *during* fine-tuning: project out activation directions corresponding to concepts the model should not use, with no change to the data or the loss. The model then generalizes differently; in their experiments this largely prevented emergent misalignment. The significance: the intervention controls not what the model does on the training data, but *which* of the many solutions consistent with that data the optimizer selects.
- **Preventative steering.** Persona vectors identify directions that control character traits: sycophancy, harmfulness, hallucination-proneness. They are used at training time in two ways. One is screening: score fine-tuning data by how far it would push the model along a trait direction, and filter it. The other is *preventative steering*: add the trait direction

during fine-tuning, so the model absorbs the training pressure without encoding the trait, then behaves normally when the vector is removed at deployment (Chen et al., 2025). Follow-on work extends the program along the pipeline in both directions. One line traces how persona directions form across pretraining checkpoints, asking when a trait becomes linearly represented and whether it could be shaped earlier (Moskvoretskii et al., 2026). Another, inoculation adapters, isolates unwanted generalization into a removable adapter module during fine-tuning, with fewer side effects than steering the whole network (Riché et al., 2026).

The through-line of regime 3 is an inversion. In the standard picture, training produces an opaque artifact and analysis begins afterward, from outside. In these methods, analysis instruments (concept directions, probes, localization maps) operate *inside the training loop*, deciding what the optimizer is allowed to learn and where it is allowed to put it. Training-time control is upstream of every after-the-fact fix: a trait that was never encoded needs no steering vector, and a capability that was localized on purpose needs no search to find.

4.5 Keeping the Signal Alive: Depth, Skip Connections, and the Residual Stream

4.5.1 Jacobian collapse

So far we presumed that gradients arrive where they are needed. Depth threatens that presumption. A deep network is a composition $f = f_L \circ \cdots \circ f_1$, and the chain rule makes the gradient at an early layer a *product* of the Jacobians of every layer above it. If the layer Jacobians have typical magnitudes below one, the product shrinks exponentially with depth; above one, it explodes. This is the **vanishing gradient** problem, and its geometric face is **Jacobian collapse**: the composed Jacobian loses rank and magnitude, and with it, the Fisher information along most parameter directions goes to zero. Training then stalls, not because the loss has a bad minimum but because the map itself has stopped transmitting.

The fix that won is a change in what a layer *is*:

$$x_{l+1} \;=\; x_l + f_l(x_l).$$

The layer no longer replaces its input. It adds a correction to it, and the uncorrected signal passes through on an identity path - a skip connection. Before this change, each layer transformed its input wholesale, and no state survived from one layer to the next except through that transformation. After it, the layer Jacobian becomes $I + \partial f_l / \partial x_l$, and a product of near-identity matrices stays near the identity instead of collapsing. The gradient now has an unobstructed route from the loss back to the first layer, and networks hundreds of layers deep become trainable (He et al., 2016).

4.5.2 What the fix quietly created: the residual stream

The interpretability consequence is larger than the optimization one. Under wholesale transformation, each layer's output is a new encoding. The network's intermediate states form a pipeline of mutually incompatible languages, and there is no one place where "what the machine currently represents" lives. Under accumulation, there is. A single d-dimensional vector

runs through the whole computation, preserved by default and edited in increments. Every layer reads the current state and writes a small addition back into the *same* coordinate system. An addition written at layer 3 must still be readable at layer 30, so the architecture forces a common language on every component. The state at any depth is the running sum of everything written so far: the machine's working notes at that moment - a latent space, not mere wiring between stages.

Two properties of that space carry the second half of this course.

First, because the vector persists across depth, a concept can persist too, as a stable direction that successive layers nudge rather than re-encode.

Second, because the edits are additive, the contribution of any single component is in principle separable by subtraction: one can ask exactly what a given layer or head wrote. Measuring representations, naming features, and editing behavior mid-computation (the projects of Part II) all operate on this one shared vector.

4.5.3 The second instability: representation collapse

The additive fix keeps the signal alive, but it turns a dial rather than removing a constraint - and the dial has a failure mode at each end. Vanishing gradients are not the only source of instability in deep residual stacks. Liu et al. (2020) traced the training failures of deep transformers to an *amplification effect*: when a layer depends heavily on its residual branch, small parameter perturbations are amplified into large disturbances of the output, and training destabilizes even where gradients flow freely. Their remedy - Admin, an adaptive initialization that starts the network near the identity map and then releases it - stabilizes early training without capping the final model's potential.

Where the normalization sits chooses which end of the dial you live on, and the choice is a tradeoff, not a solution. With Post-Norm (layer normalization applied after the residual addition), deep layers stay distinct but the amplification effect bites. With Pre-Norm (normalization inside the residual branch), training stabilizes - and a quieter pathology appears: the identity path dominates, each layer's correction shrinks relative to the stream it edits, and deep layers drift toward computing what their predecessors already computed. Hidden states become highly similar across depth; the network effectively runs shallower than its architecture. This is **representation collapse**, the opposite pole from Jacobian collapse on the same axis: too little pass-through and gradients vanish, too much and layers merge. Zhu et al. (2024) named the tradeoff the *seesaw effect*; their Hyper-Connections address it by replacing the single fixed identity path with several residual streams mixed by learned, input-dependent matrices, recovering gradient flow and layer distinctness together, and active research continues to address the same tradeoff. Part II meets the collapse pole from the measurement side: rising cross-layer similarity and falling marginal intrinsic dimension in deep layers are what it looks like in the activation geometry (Chapter 5). And the placement of normalization is a training-time lever in exactly the sense of Section 4.4: it decides, before any probe is built, how much of the network's depth there is to read.

4.5.4 Closing the arc

The trained machine does not merely reach low loss. It is selected for robust, compressed solutions, and that selection guarantees there is structure inside. But structured is not legible. The compression is optimized against the loss, not for the reader, and by default it packs meaning

into forms no unit-by-unit inspection can untangle. Sections 4.4 and 4.5 showed that the default is negotiable. Training choices set how readable and how controllable the product comes out. The newest work moves the instruments of understanding into the training loop itself. And the architecture's own signal-preserving anatomy supplies the shared workspace those instruments read and edit.

Takeaways

- "Steepest descent" is undefined until a metric is chosen; under the Fisher metric the answer is the natural gradient $\Delta\theta = -\eta F^{-1}\nabla L$ (Amari, 1998) - reparametrization-invariant, sensitivity-scaled, and equal progress per unit of distribution change.
- The full Fisher matrix is unusable at scale; K-FAC's per-layer Kronecker factorization (Martens & Grosse, 2015) made curvature affordable, and Adam's diagonal rescaling is the same idea at the crudest rung - Fisher geometry hides inside everyday optimizers.
- Flat minima generalize better (Keskar et al., 2016), but only Fisher-weighted flatness survives the reparametrization objection (Dinh et al., 2017); stochastic gradient descent's anisotropic noise plausibly drives it into Fisher-flat basins without ever computing the geometry.
- Structured is not legible: the optimizer compresses for its loss, not for the reader, and its default packing of features into shared dimensions obstructs unit-by-unit understanding. But the default is negotiable: training choices set legibility incidentally (weight decay, dropout, width), can buy it deliberately at a capability cost (activation functions, discrete bottlenecks, interpretable architectures), and can place and suppress specific concepts on purpose (gradient routing, concept-ablation fine-tuning, preventative steering) - moving the instruments of understanding inside the training loop.
- Depth threatens training through Jacobian collapse: multiplied layer Jacobians vanish or explode, zeroing Fisher information along most parameter directions. The additive fix, $x_{l+1} = x_l + f_l(x_l)$ (He et al., 2016), keeps gradients alive - and as a side effect creates a persistent, cumulative latent state, the shared workspace (Chapter 3's residual stream) on whose readability Part II depends. The strength of that pass-through is itself a dial with a failure mode at each end - instability through amplification (Liu et al., 2020) versus deep layers collapsing into redundancy (the seesaw effect; Zhu et al., 2024) - and Part II measures the collapse end in the activation geometry.

Research Directions

From §4.1: Where does natural gradient still pay?

Run matched-budget comparisons of Adam, K-FAC, and plain stochastic gradient descent on small transformer pretraining (10–100M parameters), reporting loss against both step count and wall-clock. The literature's verdict is mixed and mostly dated; a careful modern accounting on one controlled setup (including the fine-tuning and reinforcement-learning regimes where curvature methods are claimed to win) would be immediately useful, and doubles as a measurement of how much of the Fisher geometry's promised benefit normalization layers and schedule tuning already capture for free.

From §4.3: Measure the invariant sharpness of real checkpoints

Compute Fisher-weighted sharpness (via Hessian- and Fisher-vector products, tractable without forming matrices) for checkpoints along real training runs, across batch sizes, learning rates, and abrupt delayed-generalization transitions on small algorithmic tasks (the phenomenon known as grokking). Does invariant sharpness fall when generalization improves, and does it *lead* or *lag* behavioral change? A leading relationship would make it a genuine early-warning channel for capability formation - an internal observable that moves before behavior does.

From §4.2–4.3: Does the optimizer's geometry shape the learned representation?

Train identical small models with plain stochastic gradient descent, Adam, and K-FAC to matched loss, then compare their internal structure: dimensionality profiles of the activations, how many distinct input types drive each unit, and the quality of features recovered by sparse dictionary methods. Do curvature-aware optimizers produce measurably different internal structure (cleaner features, different packing), or does the data distribution dominate? Either answer matters: analyses of one model are routinely assumed to transfer to models trained with different optimizers, and nobody has checked whether that transfer is licensed.

From §4.4: Audit a legibility intervention for hiding

The Softmax Linear Units episode showed that training for a legibility metric can relocate opaque computation instead of removing it. No standard audit exists for this failure mode. Take one train-for-legibility method (a codebook bottleneck, or an activation-function change) on a small model, and measure legibility *everywhere* - not just at the intervened component: per-neuron interpretability in the MLPs, but also in normalization parameters, attention outputs, and the residual stream. The deliverable is a conservation analysis: did total opacity fall, or did it move? A reusable audit protocol here would make every regime-2 result more trustworthy, and a demonstrated hiding effect in a new setting would be a finding in its own right.

Programs for Chapter 4

`natural_vs_plain_gradient.py` Fit a Gaussian's $(\mu, \log\sigma)$ to sample data by gradient descent on negative log-likelihood, from a deliberately bad start. Run plain gradient descent and natural gradient (the 2×2 Fisher matrix is analytic - invert it exactly). Plot both trajectories in (μ, σ) space over the loss contours, plus loss-vs-step. The zigzag-and-stall versus smooth-geodesic contrast is the whole of Section 4.1 in one figure. Then relabel the dial $(\sigma \to \sigma^2)$ and rerun both: the natural-gradient trajectory through distribution space is unchanged; plain gradient's changes. Invariance, demonstrated rather than asserted. *Type: script + matplotlib. Deps: numpy only.*

`preconditioning_comparison.py` Logistic regression on synthetic data with feature scales spanning 10^3 (condition number $\sim 10^6$). Train with: plain stochastic gradient descent, Adam, diagonal-Fisher preconditioning, and exact natural gradient (feasible at this size). Plot loss-vs-step for all four. Adam and diagonal-Fisher nearly coincide (Adam *is* a diagonal Fisher method in disguise), while exact natural gradient shows what the full geometry

buys and plain descent shows life without any. *Type: script + matplotlib. Deps: numpy, torch optional.*

`flat_minima_and_noise.py` Construct a 2-D loss surface with two global minima (one wide basin, one narrow) and run gradient descent with tunable isotropic noise from many random starts. Plot the fraction of runs ending in the wide basin as a function of noise scale: near zero noise, arrivals split by basin of attraction; with noise, the narrow basin empties. Then make it geometric: define train loss and a slightly shifted test loss, and show end-of-training test loss tracks basin width. The noise-selects-flatness mechanism of §4.3, small enough to see all at once. *Type: script + matplotlib. Deps: numpy only.*

References

Papers marked with a filename in brackets are archived as PDFs in `new_book/references/`.

Chapter 1 - Statistical Foundations

1. Markov, A.A. (1913). *An example of statistical investigation of the text "Eugene Onegin" concerning the connection of samples in chains.* Bulletin of the Imperial Academy of Sciences of St. Petersburg. (English translation: Science in Context 19(4), 2006.)

2. Gamow, G. (1947). *One Two Three… Infinity: Facts and Speculations of Science.* Viking Press.

3. Shannon, C.E. (1948). *A Mathematical Theory of Communication.* Bell System Technical Journal 27. [`shannon1948_mathematical_theory_communication.pdf`]

4. Shannon, C.E. (1951). *Prediction and Entropy of Printed English.* Bell System Technical Journal 30(1).

5. Barlow, H.B. (1961). *Possible principles underlying the transformation of sensory messages.* In *Sensory Communication*, MIT Press.

6. Laughlin, S. (1981). *A simple coding procedure enhances a neuron's information capacity.* Zeitschrift für Naturforschung C 36(9–10).

Chapter 2 - Geometric Foundations

7. Nielsen, F. (2018). *An Elementary Introduction to Information Geometry.* arXiv:1808.08271. [`nielsen2018_information_geometry_intro.pdf`]

8. Chentsov, N.N. (1972). *Statistical Decision Rules and Optimal Inference.* Nauka (English translation: AMS, 1982).

9. Seung, H.S., Sompolinsky, H. (1993). *Simple models for reading neuronal population codes.* Proceedings of the National Academy of Sciences 90(22).

Chapter 3 - Neural Networks as Distribution Machines

10. Vaswani, A., Shazeer, N., Parmar, N., et al. (2017). *Attention Is All You Need.* NeurIPS 2017. arXiv:1706.03762. [`vaswani2017_attention.pdf`]

11. Elhage, N., Nanda, N., Olsson, C., et al. (2021). *A Mathematical Framework for Transformer Circuits.* Transformer Circuits Thread. `transformer-circuits.pub/2021/framework`.

12. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I. (2018). *Improving Language Understanding by Generative Pre-Training.* OpenAI technical report. [`../language_understanding_paper.pdf`]

13. Liu, P.J., Saleh, M., Pot, E., et al. (2018). *Generating Wikipedia by Summarizing Long Sequences.* ICLR 2018. arXiv:1801.10198. (The decoder-only transformer variant.)

14. Ho, J., Jain, A., Abbeel, P. (2020). *Denoising Diffusion Probabilistic Models.* NeurIPS 2020. arXiv:2006.11239. [`ho2020_denoising_diffusion.pdf`]

15. Liu, X., Gong, C., Liu, Q. (2022). *Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow.* ICLR 2023. arXiv:2209.03003. [`liu2022_rectified_flow.pdf`]

16. Shazeer, N., Mirhoseini, A., Maziarz, K., et al. (2017). *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer.* ICLR 2017. arXiv:1701.06538. [`shazeer2017_mixture_of_experts.pdf`]

17. Kanwisher, N., McDermott, J., Chun, M.M. (1997). *The fusiform face area: a module in human extrastriate cortex specialized for face perception.* Journal of Neuroscience 17(11).

Chapter 4 - Training Dynamics

18. Amari, S. (1998). *Natural Gradient Works Efficiently in Learning.* Neural Computation 10(2).

19. Martens, J. (2014). *New Insights and Perspectives on the Natural Gradient Method.* JMLR 21 (2020). arXiv:1412.1193. [`martens2014_natural_gradient_insights.pdf`]

20. Martens, J., Grosse, R. (2015). *Optimizing Neural Networks with Kronecker-factored Approximate Curvature.* ICML 2015. arXiv:1503.05671. [`martens2015_kfac.pdf`]

21. Kingma, D.P., Ba, J. (2015). *Adam: A Method for Stochastic Optimization.* ICLR 2015. arXiv:1412.6980.

22. Jordan, K., Jin, Y., Boza, V., et al. (2024). *Muon: An optimizer for hidden layers in neural networks.* `kellerjordan.github.io/posts/muon`. (Web only.)

23. Liu, J., Su, J., Yao, X., et al. (2025). *Muon is Scalable for LLM Training.* arXiv:2502.16982.

24. Shumaylov, Z., Da Costa, N., Zaika, P., et al. (2026). *Muon is Not That Special: Random or Inverted Spectra Work Just as Well.* arXiv:2605.11181.

25. Keskar, N.S., Mudigere, D., Nocedal, J., Smelyanskiy, M., Tang, P.T.P. (2016). *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima.* ICLR 2017. arXiv:1609.04836. [`keskar2016_large_batch_sharp_minima.pdf`]

26. He, K., Zhang, X., Ren, S., Sun, J. (2016). *Deep Residual Learning for Image Recognition.* CVPR 2016. arXiv:1512.03385.

27. Liu, L., Liu, X., Gao, J., Chen, W., Han, J. (2020). *Understanding the Difficulty of Training Transformers.* EMNLP 2020. arXiv:2004.08249. (The amplification effect of §4.5; the Admin initialization.)

28. Zhu, D., Huang, H., Huang, Z., Zeng, Y., Mao, Y., Wu, B., Min, Q., Zhou, X. (2024). *Hyper-Connections.* arXiv:2409.19606. (The seesaw effect between gradient vanishing and representation collapse of §4.5.)

29. Dinh, L., Pascanu, R., Bengio, S., Bengio, Y. (2017). *Sharp Minima Can Generalize For Deep Nets.* ICML 2017. arXiv:1703.04933. [`dinh2017_sharp_minima_can_generalize.pdf`]

30. Ölveczky, B.P., Andalman, A.S., Fee, M.S. (2005). *Vocal experimentation in the juvenile songbird requires a basal ganglia circuit.* PLoS Biology 3(5).

Chapter 4, §4.4 - Training-Time Levers on Interpretability

31. Elhage, N., Hume, T., Olsson, C., et al. (2022). *Toy Models of Superposition.* Transformer Circuits Thread / arXiv:2209.10652. [`elhage2022_toy_models_superposition.pdf`]

32. Nanda, N., Chan, L., Lieberum, T., Smith, J., Steinhardt, J. (2023). *Progress measures for grokking via mechanistic interpretability.* ICLR 2023. arXiv:2301.05217. [`nanda2023_grokking_progress_measures.pdf`]

33. Elhage, N., Hume, T., Olsson, C., et al. (2022). *Softmax Linear Units.* Transformer Circuits Thread, `transformer-circuits.pub/2022/solu`. (Web only.)

34. Tamkin, A., Taufeeque, M., Goodman, N.D. (2023). *Codebook Features: Sparse and Discrete Interpretability for Neural Networks.* ICML 2024. arXiv:2310.17230. [`tamkin2023_codebook_features.pdf`]

35. Hewitt, J., Thickstun, J., Manning, C.D., Liang, P. (2023). *Backpack Language Models.* ACL 2023. arXiv:2305.16765. [`hewitt2023_backpack_language_models.pdf`]

36. Jermyn, A.S., Schiefer, N., Hubinger, E. (2022). *Engineering Monosemanticity in Toy Models.* arXiv:2211.09169. [`jermyn2022_engineering_monosemanticity.pdf`]

37. Jiang, Y., Rajendran, G., Ravikumar, P., Aragam, B., Veitch, V. (2024). *On the Origins of Linear Representations in Large Language Models.* ICML 2024. arXiv:2403.03867. [`jiang2024_origins_linear_representations.pdf`]

38. Cloud, A., Goldman-Wetzler, J., Wybitul, E., Miller, J., Turner, A.M. (2024). *Gradient Routing: Masking Gradients to Localize Computation in Neural Networks.* arXiv:2410.04332. [`cloud2024_gradient_routing.pdf`]

39. Casademunt, H., Juang, C., Karvonen, A., Marks, S., Rajamanoharan, S., Nanda, N. (2025). *Steering Out-of-Distribution Generalization with Concept Ablation Fine-Tuning.* arXiv:2507.16795. [`casademunt2025_concept_ablation_finetuning.pdf`]

40. Chen, R., Arditi, A., Sleight, H., Evans, O., Lindsey, J. (2025). *Persona Vectors: Monitoring and Controlling Character Traits in Language Models.* arXiv:2507.21509. [`chen2025_persona_vectors.pdf`]

41. Moskvoretskii, V., Glandorf, D., Medina Moreira, J., Käser, T., West, R. (2026). *Tracing Persona Vectors Through LLM Pretraining.* arXiv:2605.13329. [`moskvoretskii2026_tracing_persona_vectors.pdf`]

42. Riché, M., Tan, D., Kohonen, V., Warncke, N., et al. (2026). *Inoculation Adapters: Improved Selective Generalization of Capabilities with Fewer Surprising Backdoors.* arXiv:2606.30252. [riche2026_inoculation_adapters.pdf]

43. Betley, J., Tan, D., Warncke, N., et al. (2025). *Emergent Misalignment: Narrow fine-tuning can produce broadly misaligned LLMs.* arXiv:2502.17424. [betley2025_emergent_misalignment.pdf]

Continued in Part II

44. The references for Chapters 5–8 (representation geometry, sparse autoencoders, circuits, steering, faithfulness, and the neuroscience lineage) are collected in Part II (`new_book/book/`), with archived PDFs indexed in `new_book/references/README.md`.