2 **Optimization**: compute predictions (at current prediction) → compute the loss → backpropagate the error (which direction to change) → update parameters → zero the gradients. (Anatomy of training steps)

⇒ **math notation**: given data $D\langle X, Y\rangle$, ② model $M: \theta, Y \to \Delta(Y)$, ③ loss function $L(\theta; X; Y) = -\log P_M(\theta; X)(Y)$. Find params that minimise loss for data: $\hat{\theta} = \arg\max_{\theta \in \Theta} \sum_{x,y \in D} L(\theta, x, y)$

⇒ **loss function**: quantify the difference.
1/ MSE: $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. 2/ Cross-entropy loss: $-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij}\log(\hat{y}_{ij})$.

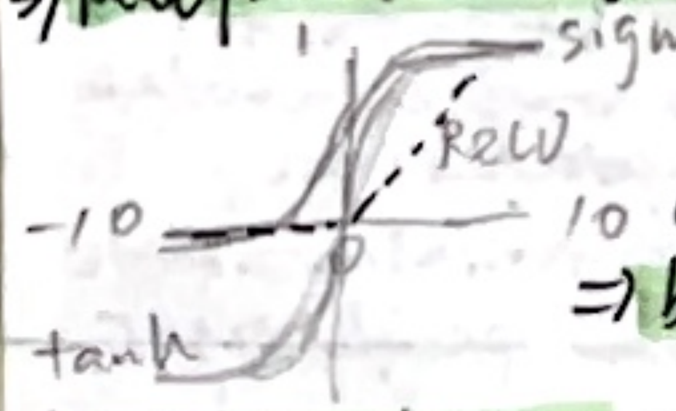
⇒ **Gradient**: vectors of partial derivatives of the loss function which suggests the rate & direction to update params (opposite to gradient, <0 increase, >0 decrease) in order to minimise loss function ($\frac{\partial L}{\partial V}$ math)

⇒ **Back-propagation**: calculate gradients by propagating the errors backward through the net from output to input, updating each param along way.

⇒ **Stochastic GD**: use random subsets (mini-batches) of data instead of the entire dataset to iteratively calculating loss, gradients & update params. (vanilla / Batch GD)

ANN ⇒ **Unit score**: $z = b + \sum_{j=1}^{n} w_j x_j = b + w \cdot x$ → input
x_1, x_2, x_3 → w_1, w_2, w_3 → (Σ) → z → [σ] → y   ⇒ ANN/MLP: $a = \sigma(z) = \sigma(Wx + b)$ (output, activation func, bias)

⇒ **activation function**: Perception: $f(z) = \delta z > 0$.
1/ **Sigmoid** (0,1): $f(z) = \sigma(z) = \frac{1}{1+\exp(-z)}$
2/ **Hyperbolic tangent** (−1, 1): $f(z) = \tanh(z) = \frac{\exp(z) - \exp(-z)}{\exp(z) + \exp(-z)}$.
3/ **Rectified linear Unit**: $f(z) = ReLU(z) = \max(z, 0)$. 则z是否正



⇒ **weights**: params that control strength of connections between neuron
⇒ **bias**: param that shift activation func, allowing models to fit data better.

⇒ **hidden layers**: intermediate layers that performs computations.
⇒ **score**: last layer output, often raw score or logits (that could transform to prob.) (未经 NL 处理的数据 / 不经过 NL 的线性分类器 model)

LM ⇒ **Concept**: a function that assigns to each input X a probability distribution over S given param $\theta \in \Theta$ (input conditions; $LM_\theta: X \mapsto \Delta(S)$; sequences of tokens)
⇒ **Neural LM**: an LM realised as a neural network.
⇒ **Causal LM**: a function that maps an initial token sequence to a next-token distribution.
LM: $w_{1:n} \mapsto \Delta(\mathcal{V})$ $\mathcal{V}$ for vocab. set of tokens.

⇒ **Surprisal**: $-\log P_{LM}(w_{n+1} \mid w_{1:n})$ next-token prob (minimize)
⇒ **Goodness of fit**: 1/ **Perplexity** $P_{LM}(w_{1:n}) = P_{LM}(w_{1:n})^{-\frac{1}{n}}$ (minimise)
2/ **Avg. surprisal**: log of PP: $= -\frac{1}{n}\log P_{LM}(w_{1:n})$.

⇒ **Auto-generation**: generate data based on previous elements, left-to-right, no bidirectional. (inputs → embeddings – RNN – Softmax – sampled word) maintain a hidden state that capture contexts

⇒ **Decoding scheme**: 1/ **pure sampling**: pure NTP. 2/ **greedy decoding**: best word (highest NTP). 3/ **Softmax**: softmax of NTP. 4/ **top-k sampling**: k most likely words. 5/ **top-p** ~: at least NTP p. 6/ **Beam Search**: top beam w/ etc.

⇒ **Auto-regressive LM**: one that only have access to previous tokens (and output beyond inputs)

⇒ **Masked LM**: predicts missing tokens in a sequence by masking some tokens and train the model to fill in the blank. Eval: downstream NLP tasks.

⇒ **Bidirectional** ~: predicts tokens based on preceding and succeeding context in a sequence. Eval: all etc.

⇒ **Training regimes**: 1/ **teacher forcing**: LM is fed true word seq. not predictions. 2/ **autoreg** (i.e. free running mode): auto-regly generates a seq. 3/ **Curriculum** ~: hybrid of both. 4/ **Professor forcing**: 1 + adversarial. 5/ **Decoding-based**: used prediction function (decoding scheme) to optimise based on actual output.

RNN: NN designed to process sequential data where the output depends on the current & past input
⇒ **FFNN**: info flow in 1D. ⇒ **Convolutional NN**: specialised for processing grid-like data (图像).
⇒ 5 vanilla FFNN, RNN: capture more context while avoiding sparsity, storage, compute issues. Hidden layers represent word meaning.
⇒ Instead of every time step, do every T step (分段处理). 但不是 window size 固定 而是 infinite memory.
⇒ 优势劣势: **Strength** 1/ can handle infinite-length seq. (not just fixed window) 2/ has a memory of the context (hidden layers recurrent loop) 3/ same w & b for all input, so positionality isn't overwritten (不同于 FFNN).
Issues: 1/ BPTT slow & expensive. 2/ vanishing/exploding gradient (大小幅度 无法建立 太长期 context, recency bias).
⇒ * **word-embedding**: type-based repre. **contextualized** ~ token- (models word vec; model LSTM/RNN)

LSTM: type of RNN that is capable of learning long-range dependencies (long-short-term-memory)
⇒ **Intuition**: At each time step t, we have a hidden state h^t, and cell state C^t (to store 长 info). LSTM erases, writes, and reads info into from the cell. (C^t never undergoes a nonlinear activation, just − and +. (+ of 2 things doesn't modify the gradient, just add grad). Unlike RNN which run both LTM STM through a single path, LSTM use 2 seperate path for each.
⇒ **Three gates** to regulate info flow: 1/ input ~: what new info to store in cell. 2/ forget gate: what info discarded. 3/ output ~: what part of cell state is used to compute output (the new STM).

⇒ 优劣: 优/ captures long-range dependencies well 2/ 打败 RNN. 劣 1/ more weights to learn, so 2/ higher training data demand 3/ (still) suffer from vanishing/exploding gradients.

Bi-LSTM ⇒ use both 左到右, 右到左 context.
⇒ 优劣: 优/ 多方面 单向 无视 一样的. 劣/ slower to train. 2/ 需 full data access.

Stacked-LSTM: use multiple layers of LSTM cells stacked on top of each other, these stacking hidden layers provides increased abstractions (hold meaning) (BiLSTM 是 种 Stacked)

ELMo: Goal: to obtain highly rich embeddings for each w, use both bi-directional & stacked, linearly combine all abstract repe (h) and optimise w.r.t particular tasks – 2 + yields very good contextualised embedding. ⇒ Main Takeaway: give enough info & data, having tons of explicit connections between w vectors is useful. (bi-LSTM 无 ... embeddings for words)

ATTENTION: (mechanism that compute relationships between S or inps.)
⇒ **Intuition**: at each step the decoder pays attention to all of the encoder's hidden states.
⇒ **Self-Attention**: Within same seq., each word is transformed into a rich, abstract repr. (context emb~) based on the weighted sums of other words. (其注意 how much should each word be influenced by its neighbors, the explicit positionality).
⇒ 优劣: 优 1/ context-aware 2/ more powerful than static embeddings. 劣 1/ slower to train 2/ 需 computed 计算

Transformers: 模型 NN that use self-att mechanisms for recurrent layers to process data.
⇒ **Encoder**: component that receive inputs. It produces contextualised embeddings r_i for each word x_i.
⇒ **Decoders**: ~ produce output. It generates new seq. of text. They are identical to encoders, 除 they have an additional att head in b/w the self att and FFNN layers, this focuses on parts of the encoder's repr.
⇒ **BERT** (Bidirectional En Repre from Trans): Bidirectional, only trans encoder, no decoder.
⇒ 2 training objectives: 1/ Predict the masked word. 2/ 2 sentences are fed in at a time. Predict if 2nd follows 1st.
⇒ 优劣: 优 / great for embedding learning 2/ powerful to transfer learning to other task. 劣 no deco, not easily generative

autoreg / left2right vs Bidirectional
1/ computation for input $x_1..x_3$ blind / sees x_4, x_5 to.
2/ $y_1/y_1..y_5$ is embedding for input $x_1..x_5$.
3/ y_3/y_i is a left/bidirect. contextual embeddings.

⇒ **Kind of LLMs**: 1/ **core-LLMs** (foundation models): predict statistically likely next token (GPT2, LLaMA2...). 2/ **Prepped LM** (assistants): fine-tuned, predict to please user (GPT3, LLaMA3 Instruct). 3/ **LLM-based App (agents)**: algorithm using LLM (sophisticated prompting, ChatGPT w/ tools, AutoGPT).

ICL (情境 In-context L) LLM respond to & learn from the context.
→ **Test 4 factors**: 1/ correctness of input-label mapping 2/ Similarity of examples and target input 3/ coverage of label space in examples 4/ format of input label example. (matters little)
→ method: zero-shot, k-shot
→ **Summary**: 1/ ICL need not be genuine "learning" as structure of simple prompt also matters. 2/ K-shot examples may provide statistics which LLM can exploit.

4

3/ similarity of demonstrations and target matters. 4/ instructions and label correctness do not matter, models rely more on structure of prompts. 4/ Still good explanation for labels & prompts helps.

=> Chain of Thought P: give task instruction, give ≥1 examples. (step-by-step reasoning).

=> Zero-shot P: Provide answer without any example / additional context.

=> Self-consistency P: few-shot CoT P + self-generated CoT sequence (generates multiple answers and 选择 most consistent one). ensemble

=> Knowledge Generated P: use generated by knowledge (K) for Q to generate several A to Q for each K, final answer to Q is max of weight A's.

=> Tree of Thought => 4步: 1/ thought decomposition. 2/ thought generation. 3/ state eval 4/ Search algorithm.

=> Few-shot P: provides a few examples to guide response.

5 Param Efficient FT 1/ Selective: FT only a subset of initial LLM params.

2/ Reparameterization: FT model weights using a low-rank representation.

→ LowRankAdaption: inject 2 rank decomposition matrices

3/ Additive: add trainable layers or params to model.

→ QuantizedLoRA: extend LoRA by applying quantization techniques to add

→ full FT: adjust all params. → prompt Tune: only a few add trainable "soft-prompt" to inputs

Supervised FT: further trained a pre-train model on a specific and labelled dataset to perform on a specific task.

→ Pretrained LM: a model trained on a larger dataset and can be used as a starting point for further training.

=> Instruction FT: each prompt/completion pair incl. a specific "instruction" to the LLM. vs Conventional FT: it included language instructions paired with appropriate response. while the later is without explicit instructions in the input data.

The task of RL is to solve MDP such that the expected return is maximized & find the optimal policy. The central framework for formalizing RL problems are Markov Decision Processes:

→ 组成部分: State, Action, Transition, Rewards.

=> Policy-Gradient Methods: approach to solve MDP.

→ Goal: find optimal θ. max E π θ [G t]

→ idea: optimise policy directly, such that expected reward is max.

→ Now gradient ascent: θnew = θold + α∇θ

∇J(θ) = Σ P(τ,θ) R(τ).

RLHF

=> Reward modeling: a tech in RL. Reward function that operationalises the Agent's goal (helpful, honest, harmless). Defines what to do, not how.

=> Learning from Human: use human judgements as a signal on what model predictions counts as a good output. But human slow, so learn a reward model representing human feedback.

→ online learning: 实时反馈并更新. offline: 训练完后使用.

→ Human feedback types: comparison of 2 选项 / ranking / scalar / textual / correction / label.

→ RLHF steps: 1/ collect demonstration data and train a supervised policy using labelled examples to show desired behaviour. 2/ Collect comparison data by ranking model outputs, then train a reward model to evaluate the quality of outputs 3/ Optimise the policy with reinforcement learning from feedback

Supervised FT: use labelled data to adjust model for specific task.

Unsupervised: ... unlabelled ... often through self-supervised learning tech.

Instruction-tuned: using NL instruction

6 => Core LLM vs Prepped LLM. 先 pretrained, LMing objective, 后 fine-tuned model. usefulness objectives. (extension / specialised adaption to 任务). FT version of the core

RAG Retrieval Augmented Generation: use a pre-built database of knowledge. (information retrieval) during generation.

→ Motivation: key difference compare to non-RAG LLM: the model can utilise the user's query & the relevant data at the same time, rather than relying solely on the user input & already trained data.

Agents. an autonomous entity that interacts with its environ to perform task / achieve goals using LLM

=> Tool-using Agents: LLM can learn / be FT to use external tools 如计算器, 搜索引擎, API etc. → General Approach: FT pre-trained LM on dataset that has text and API calls with there results

→ Steps: LM dataset → sample API calls → Execute API calls

→ Filter API calls → LM dataset with API calls

- LM as knowledge base: has common sense, encode human 知识

- LM as planner: LLM can produce structured action sequence for a given goal, but issue is map them onto a set of primitive actions that we can execute. Solution: generate next action in sequence / use compare similarity of generation to list of executable actions (embedding, BERT) / insert text of known action in LLM / loop.

=> Autonomous Agents: a system situated within an environ that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future. Early steps: AutoGPT. BabyGPT. JARVIS / HuggingGPT.

=> Generative Agents: capable of generating outputs that mimic human creativity or problem-solving abilities.

→ Simulacra: implies models that simulate reality or human-like behavior, often blurring the line between what is real & artificial.

=> TESCREAL: 1/ Transhumanism: use tech to engineer better human race; 2/ Extropianism: reach longevity, possibly defeat death. 3/ Singularitarianism: belief in / strive for singularity. 4/ Cosmism: human life expanding into space. 5/ Rationalism: reason and principled argument as sole source of knowledge / decision making. 6/ Effective Altruism: rational deliberation on how to do the most good given available resources. 7/ Longtermism: no discounting factor on future utility, what matters is the far-future maximization of intelligent life.

=> Langchain: framework designed to facilitate devpt of apps using LLMs.

=> Cognitive Model: a model that integrates elements of NN with symbolic reasoning. Aims to combine strength of both paradigms: NN's ability to learn from 大量 data, structured, interpretable repr of symbolic AI.

7 Debugging. why? Models are complicated & opaque, everything is hyperparams, no global optimisation.

→ Possible causes: 1/ Training time problems: lack of model capacity, poor training algorithms, training time bug. 2/ Test time problems: disconnect between training and test, failure of search algorithms. 3/ overfitting. 4/ Mismatch between optimized function and eval. (最好一个一个地 debug, 逐步排除)

→ Optimisation issue: check your optimizer, lr, initialisation (variables), minibatching. Documentation helps.

→ Debug minibatches: 先确认 calculate loss with larger batch size, 再 calculate loss for each sent individually and sum 这两个值应该一致.

→ Debugging structure generation: call decoding function / loss function, 输出应该一致.

→ Beam search: 不只 pick one high-prob word, maintain several paths.

→ N search: better search, better model score.

当用 maximum likelihood 来训练, 但是改进 ... better, loss going down, 但是 acc 变 worse

如何解决 Debugging? Best Practices. ① Fiddle with inputs

=> what-if scenarios. use model to explore hypothetical scenarios. Ask model to generate responses based on 不同 "what if" questions to predict outcomes.

=> Shapley value → Intuition: a fair way to attribute the total gain to the players (in proportion to their contribution)

→ Notation: the avg. marginal contribution across all permutations.

→ 优点: 1/ sound theoretical foundation. 2/ model diagnostic (no rely on particular params / structures) 3/ directional (正负)

缺点 1/ Factorial complexity 2/ no mechanistic understanding

Having attention weights (of attention head)

=> Attention visualisation: abnormal attention weights

=> Gradient Tracing: Naive → Intergrate Gradients: integrate the gradients along a straight-line path from baseline to input

2/ Integrated Gradients explains F(input) − F(baseline) in terms of input features. 例如 CAM (Class Activation Mapping) 来 build.

→ 优点: 1/ model specific 2/ applicate to complex models.

缺点 1/ input dependence. 2/ Gradient saturation.

=> Probing: a technique to analyse NN by training simple classification to determine if they encode specific ling or task-related properties.

→ Motiv/ 目的: 1/ Classifier trained predicts property: property encoded, 2/ ~cannot~: not encoded or not encoded in a useful way.

→ Probing sensitivity: measures the probe model's ability to make output decisions independent of ling properties of the repr.

=> Knowledge Editing: add extra contexts to a token.

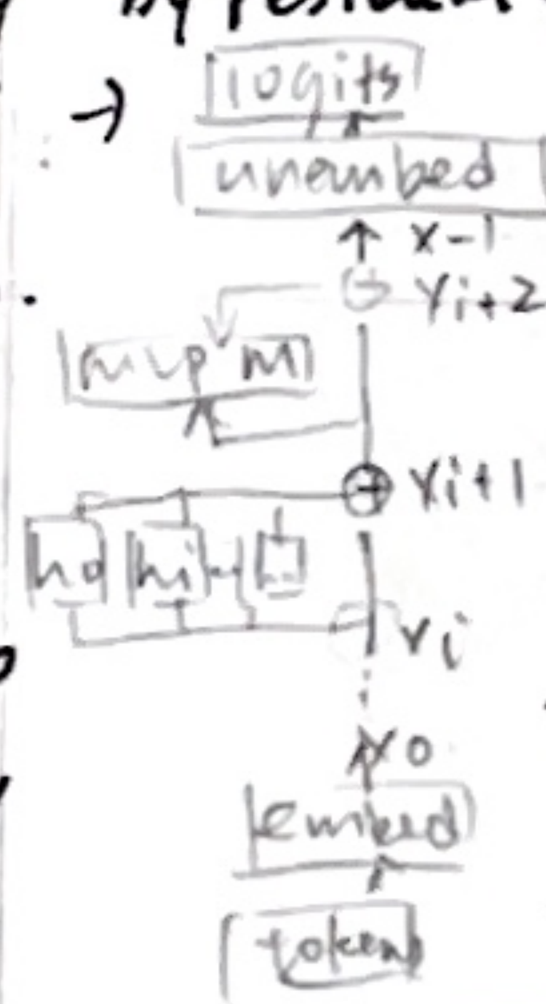
2/ 一个例子:

→ RNN: weights & bias 不变, shared across every input no matter how many times unrolled.

→ RNN issue: vanishing / exploding gradient, thus LSTM.

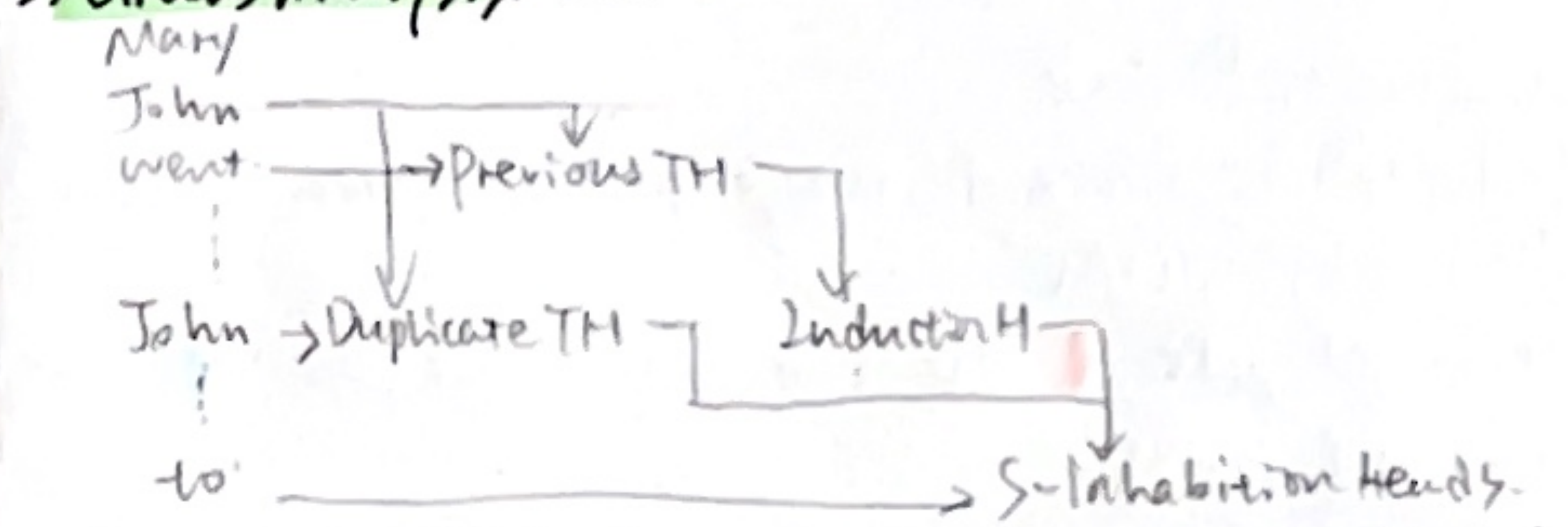
8 => Counterwise of behavioral assmt: test behavior of S under the exact same circumstances (in application context C).
(vs goldstandard) => tale-telling example: 4 good for motivation or suggestive demonstration => strong conclusion only if a single example is evidence enough, and backup by more systematic testing.
=> Benchmark testing -> What? standardised set of tasks or datasets with fixed standards of eval. -> What for? Systematic assmt of capability, streamlined comparison of different systems, identify w/strengths etc.
-> common collection (GLUE, SuperGLUE, BIG-BENCH.
-> BigBench: JSON (multi-choice / classification, generation) programming task: direct interact with model by python codes.
=> Eval metrics => accuracy (argmax): raw score:
$s(y_{ij}) = \sum_{(z=0)}^{(y_{ij})} \log P(y_{ij} \mid x_i)$ -> input prompt; answer option: y_{ij}; item $i \in I$; how probable is a particular answer
argmax: $c_i = \arg\max S(y_{ij})$ (true option). acc: $E_i \delta_{c_i} = t_i$ (E for Expectation)
=> softmax acc (stochastic choice): $P(c_i = j) \propto \exp(\alpha\, s(y_{ij}))$
acc (now a function of α): $E_i E_{c_i} \delta_{c_i} = t_i$
=> Confidence calibration: idea: predict category prob should approximate true probabilities (即模型预测的概率应与实际的准确率相符, 用calibration matrix 来看)
-> Expected calibration error (proportion of that p = p is the [illegible])
-> even acc is high, they could still be overconfident about [illegible]
-> calibration improves as model are made larger
=> Bias correction. ① length correction: achieves better ② prompt variation (properties), from prompt format / training examples / order of example, resulting in majority label bias (prefer more freq.), recency label bias (later), common token bias (common). ③ unequal numbers of (near) synonyms.
(vs human) Machine Psychology: treat LM as a participant in a psychological experiment.
=> vs Benchmark: the later has gold standard, the other more interested in human likeness. => the later goes into large coverage in order to get general info on LM as a class, the former is more in details, whether LM can do certain things either in general or like humans do.
=> What for: 1/ systematic assesst of capability (能力) 2/ studying ML-system as a new natural kind (which human-relevant) capa are present? 3/ using experiment method inspired from 行为科学.
=> Semantic illusion: Q含disguised error, 旨在激发intuitive but incorrect system response.
=> Target Assmt: experimental-based method, test sent pairs with focusing on 3 现象: 1/ subject-verb agreement, 2/ reflexive anaphora 3/ neg polarity.

-> Define grammatical prediction: an LM is said to predict the right grammaticality judgment iff:
$P_M(w_{1:n}) > P_M(v_{1:m})$
-> a more subtle score: Syntactic generalization (SG) score. e.g. "garden-path sent". 从不同starting-point 算 NTP.
-> 关键变量: careful experimental design. => training set size 重要, 但 model archi 更重要.
=> Assessing (Human) Lang Processing (处理) (结合人类中间 [illegible])
=> Surprisal theory (to measure sent 难度): $-\log P_M(w_i \mid w_{1:i-1})$
{ prediction: actively predict
{ integration: passive preactivating lead to some than other
- Empirical evid: cloze prob. eye tracking reading, self-paced reading, maze task...
- the higher the log-likelihood (ΔLL), the better. model 对 data 的预测越准.
=> ML Best Practices: clear RQ/hypothesis; severe testing, not confirmation seeking; preregister analyses; no overselling of results; tracing contamination [illegible]
9 Mechanistic Interpretability: 旨在解释中间层次的信息表达方式.
=> Early Decoding / logit lens: project intermediate tokens to high-dimen representation of a Transformer model back to the vocab space early in the net to interpret what initial layers want to say. 提前揭示模型中间层 [illegible] 词汇 [illegible]
-> 颜色中, the whiter, the more certain.
=> Residual Stream: input & output of each layer being added by residual connections to keep parts of original input.
-> final logit: by applying the unemb.



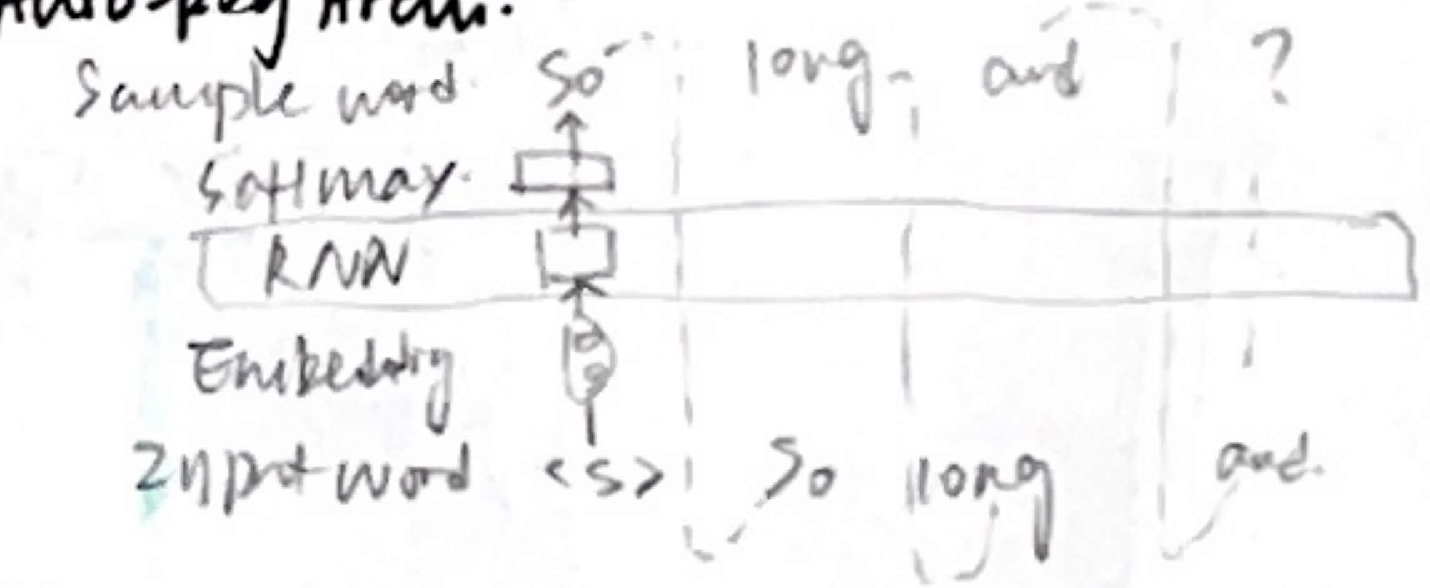
} An MLP layer m, run and added to the resi str.
$x_{i+2} = x_{i+1} + m(x_{i+1})$.
} Each attention head h_i ...
$x_{i+1} = x_i + \sum h(x_i)$.
) -> residual block
-> The res-str is modified by a seq. of MLP and att layers "reading from" the stream and "writing to" it by adding a linear transformation.
=> Attention head (info-management) ① read & write 到残差流, 3不同位 ② 定 update connections cross token boundary. ③ linear subspace relating to other tokens emerge in "contextual word embeddings". (是在W subspace 一致 [illegible], 从A位, 从B token 写入)
-> 有 2 key operations: ① Output-value (OV) circuit: determines how a given token affects logit. ② Query-value (QV) circuit: controls which token the head prefers.
-> Induction head: 上下文出现过的token, 该头 attend to 预测 next token; unseen: focus on first token in the seq.

* 体系: bottleneck Activation: 不同层的信息, 受限于 [illegible]
=> Activation Patching: 将以前层激活值 新 [illegible] 修改 [illegible] 到 [illegible]. 从而 搞清 model which part data 负责 search 和 store 相关 data (fine the)
-> Expressive Metric: logit Difference: b/w (不)正确 answers. Raw class logit: for the c/a. LogProb: ⌐c/a. RawProb: c/a. Binary performance metrics: such as acc/rank of c/a.
-> Patch Direction: 1/ clean-to corrupted patching: shows whether activations are sufficient to restore model (行为) 2/ corrupted to clean: necessary to maintain the behav (引入噪音 noising)
=> Circuit Analysis: more heads.

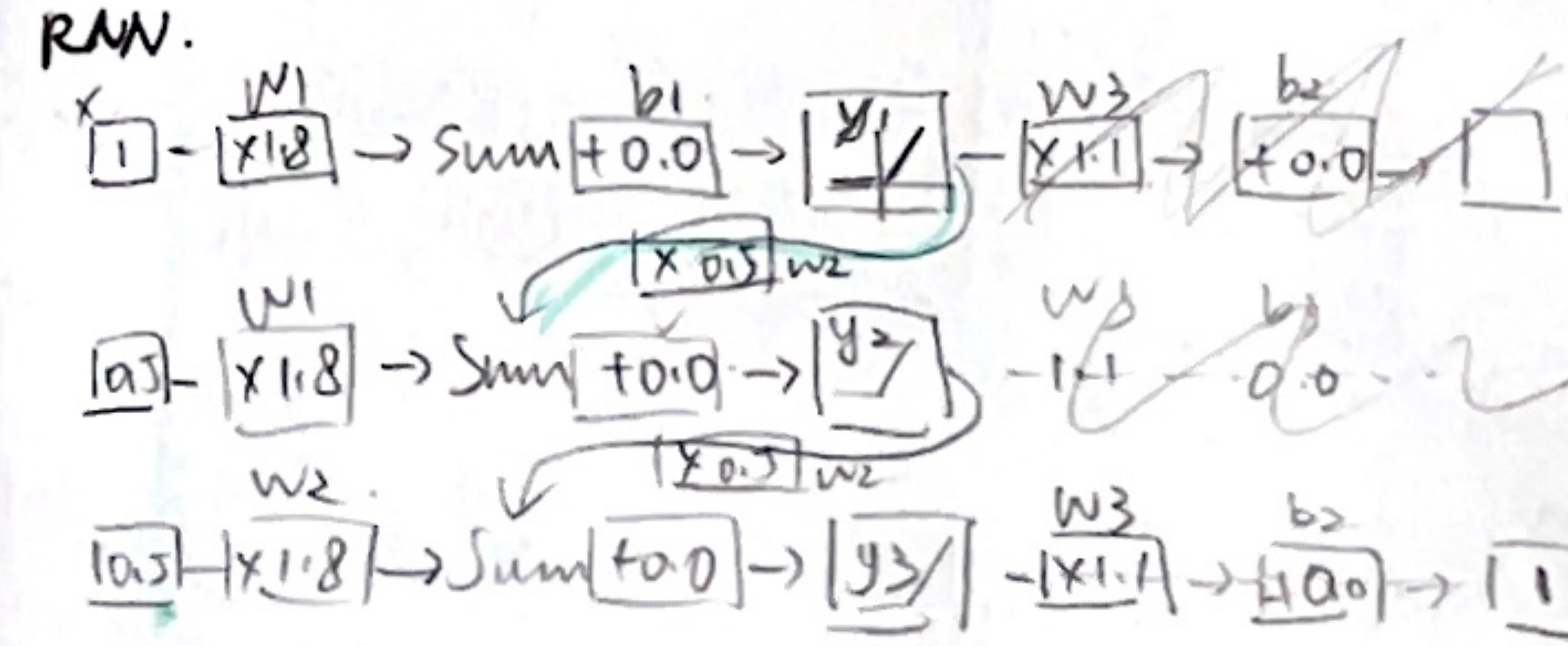


Causal scrubbing: systematically test hypothesis by recursively find all invariant model components.

Auto-reg Archi.

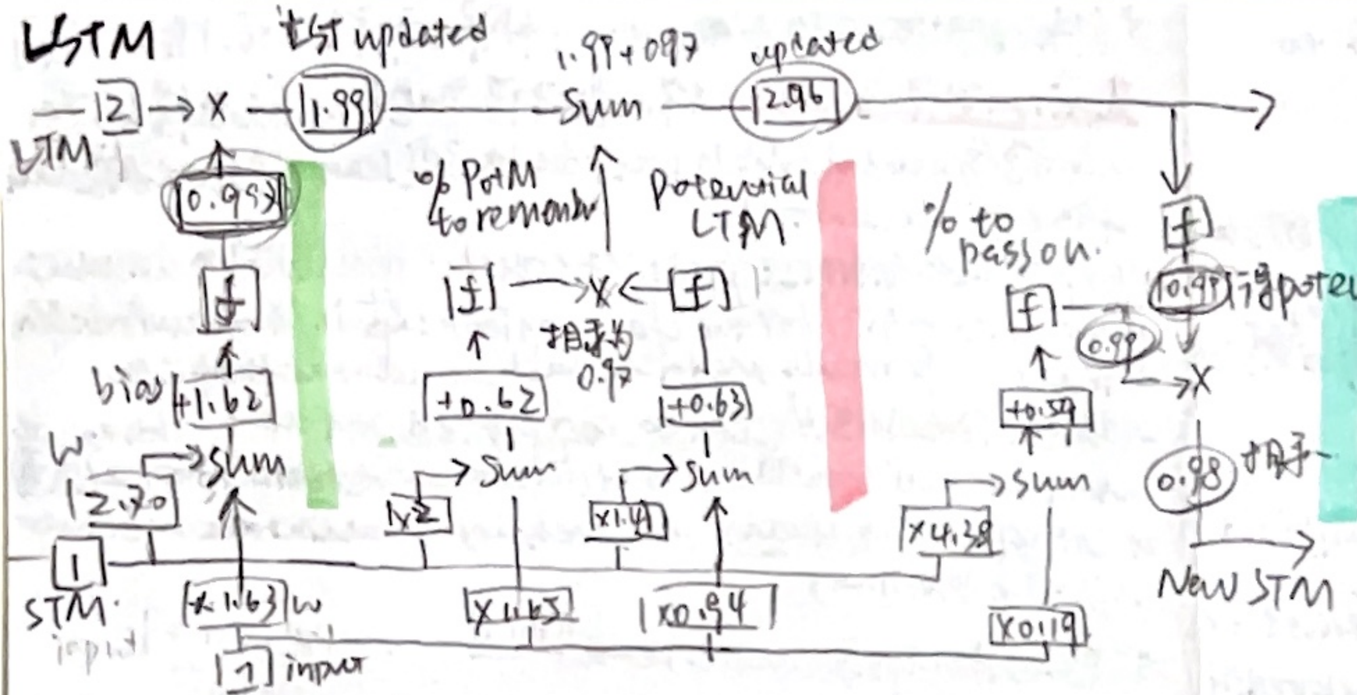


RNN.



$z = x_1 \cdot w_1 + b_1 = 0 \times 1.8 + 0.0 = 0$ => x-axis
$f(x) = \max(0, x) = 0 = y_1$ => y-axis
$(x_1 \cdot w_1) + (y_1 \cdot w_2) + b_1 = 0$ => x.
$f(x) = 0$. y_2
$y_3 = y_3 \times w_3 + b_2$

LSTM



$z = (STM_1 \cdot 其w + X_1 \cdot 其w) + b$

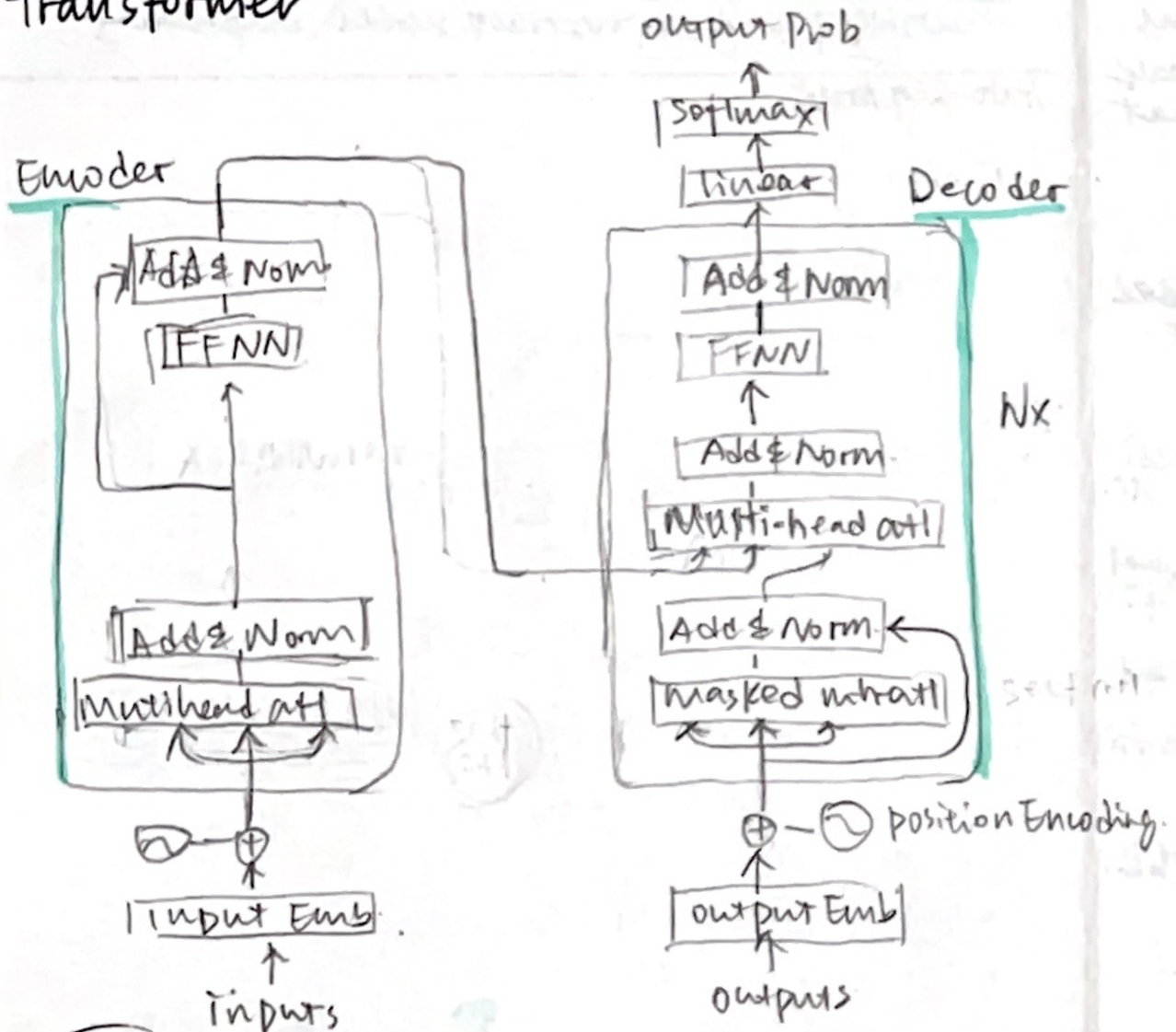
pass to $\sigma(w)$ 得到 $y = 0.99$ 之代表what % of LTS is remembered.

再 LTM·y 得到新的 LTM

the forget gate　the input gate (先看左，再右)　output Gate

关 2.96 pass to f → 0.99

Transformer



add: residual connections (adding prev. to next.) and then apply normalization

LoRA 在 self-att 前，input embeds into.