NPFL140 Large Language Models
LLM Training

http://ufal.cz/courses/npf140

Ondřej Dušek

17.3.2025

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Training a neural language model the basic way

- Reproduce sentences from data
  - **replicate exact word at each position**
  - always only **one next word**, not the whole text in one
- Fully trained from data
  - initialize model with random parameters
  - input example: didn't hit the right word → update parameters

reference: *Blue Spice is expensive*

- Very **low level**, no concept of sentence / text / aim

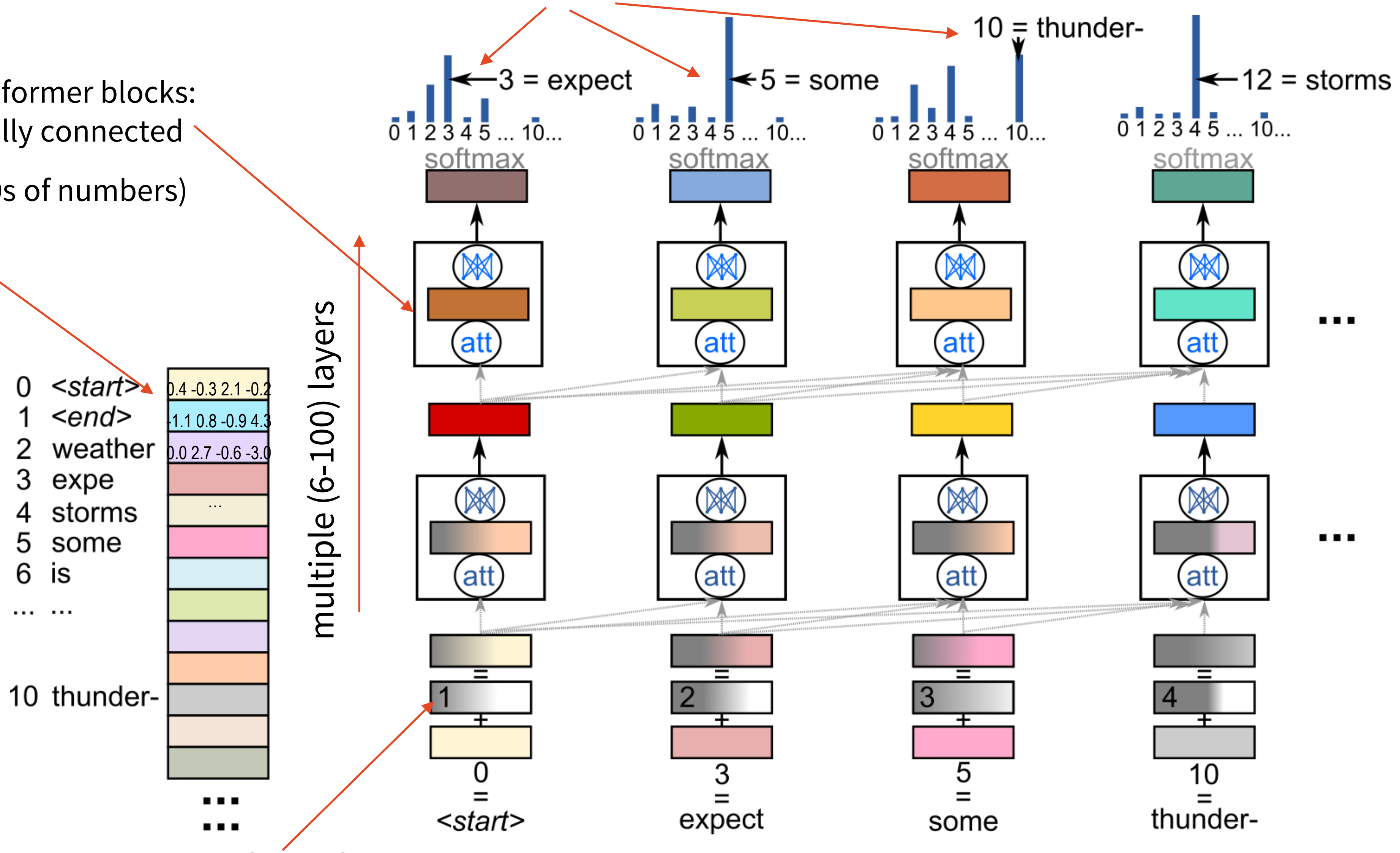
Training Transformers

in parallel: feed in training data & try to **predict 1 next token at each position,** incur loss

layers – Transformer blocks:
attention & fully connected

embeddings (~100s of numbers)

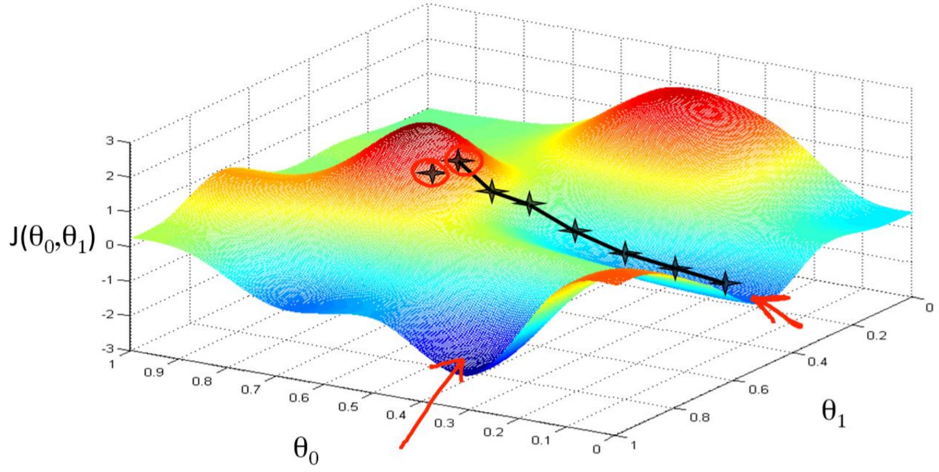
numbered
subwords



positional encoding

Gradient Descent

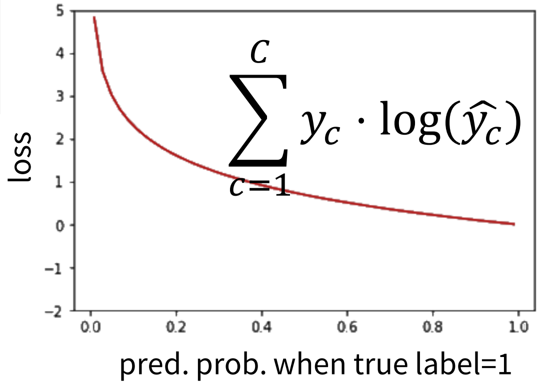
- any neural net (supervised) training– **gradient descent** methods
  - minimizing a **cost/loss function**
    (notion of error – given a model output, how far off are we?)
  - calculus: derivative = steepness/slope
  - **backpropagation**: derivatives of all parameters w. r. t. cost (compound function)
  - follow the slope to find the minimum – derivative gives the direction
  - **learning rate** = how fast we go (needs to be tuned)
- gradient averaged over **mini-batches**
  - random bunches of a few training instances
  - not as erratic as using just 1 instance,
    not as slow as computing over whole data
  - **stochastic gradient descent**



https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

Cost/Loss Functions

- differ based on what we're trying to predict
- **default: logistic / log loss** ("cross entropy")
  - for any classification / softmax – including **word prediction** in LMs
    - classes from the whole dictionary
    - correct class has <100% prob. → loss is >0
  - pretty stupid for sequences, but works
    - sequence shifted by 1 ⇒ everything wrong
- other options:
  - squared error loss – for regression (floats)
  - hinge loss – binary classification (SVMs), ranking
  - many others, variants

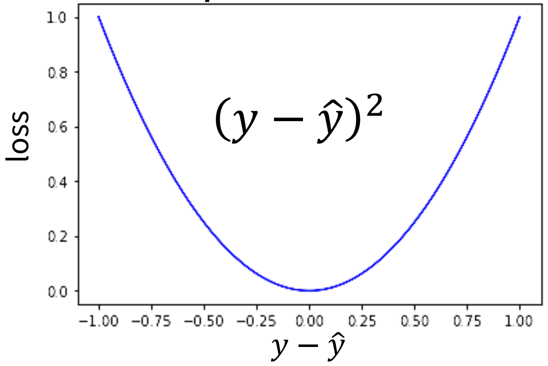


logistic

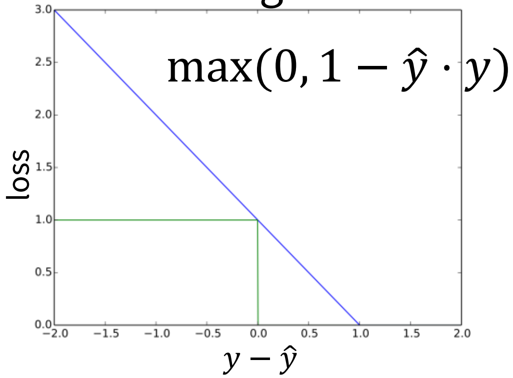
reference: *Blue Spice is expensive*

prediction: *expensive*
cheap
pricey
in the expensive price range

squared error



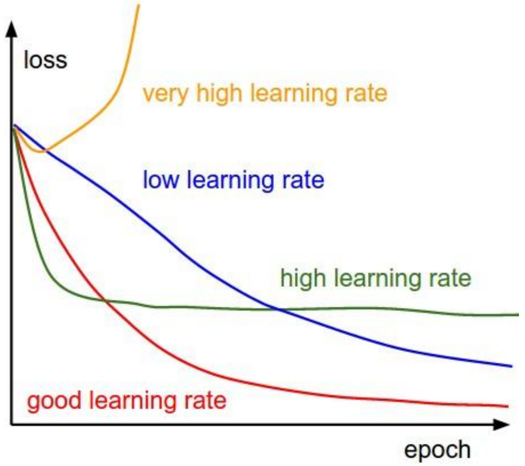
hinge



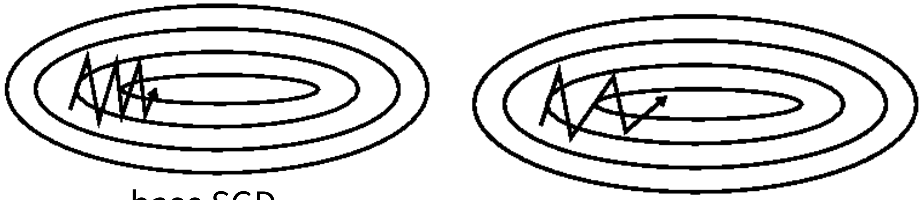
https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/
https://medium.com/@risingdeveloper/visualization-of-some-loss-functions-for-deep-learning-with-tensorflow-9f60be9d09f9 , https://en.wikipedia.org/wiki/Hinge_loss

Learning Rate & Momentum

- **LR: most important parameter** in (stochastic) gradient descent
- tricky to tune:
  - too high LR = may not find optimum
  - too low LR = may take forever
- **Learning rate decay**: start high, lower LR gradually
  - make bigger steps (to speed learning)
  - slow down when you're almost there (to avoid overshooting)
  - linear, stepwise, exponential, reduce-on-plateau…
- **Momentum**: moving average of gradients
  - make learning less erratic
  - $m = \beta \cdot m + (1 - \beta) \cdot \Delta$, update by m instead of Δ



http://cs231n.github.io/neural-networks-3/



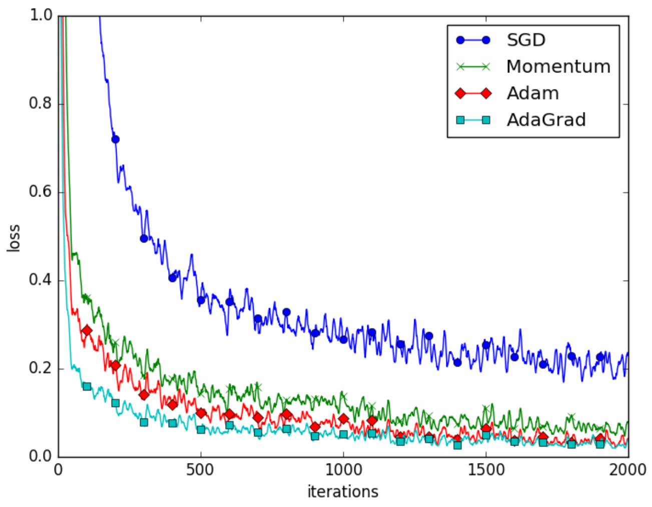
base SGD

momentum

https://ruder.io/optimizing-gradient-descent/

Optimizers

http://kaeken.hatenablog.com/entry/2016/11/10/203151

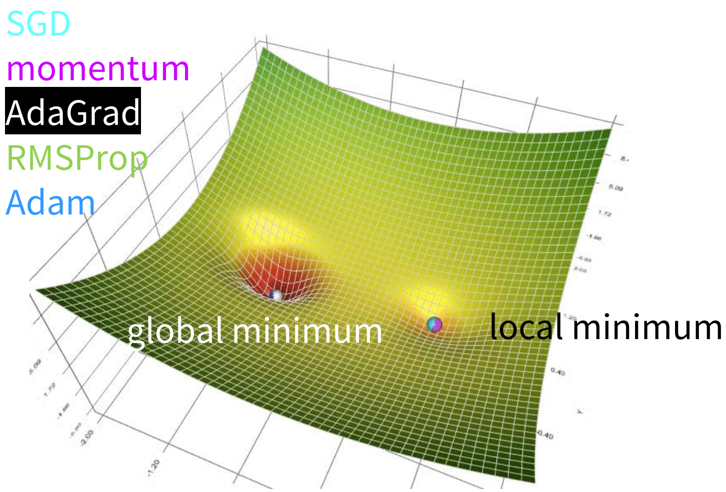


- Better LR management
  - change LR based on gradients, less sensitive to settings
- **AdaGrad** – all history
  - remember sum of total gradients squared: $\sum_t \Delta_t^2$
  - divide LR by $\sqrt{\sum \Delta_t^2}$
  - variants: **Adadelta**, **RMSProp** – slower LR drop
- **Adam** – per-parameter momentum
  - moving averages for Δ & Δ^2:

    $m = \beta_1 \cdot m + (1 - \beta_1)\Delta,\ v = \beta_2 \cdot v + (1 - \beta_2)\Delta^2$
  - use m instead of Δ, divide LR by $\sqrt{v}$
  - often used as default nowadays
  - variant: **AdamW** – better regularization
    - not much difference though

(Kingma & Ba, 2015)
https://arxiv.org/abs/1412.6980

(Loshchilov & Hutter, 2019)
https://arxiv.org/abs/1711.05101

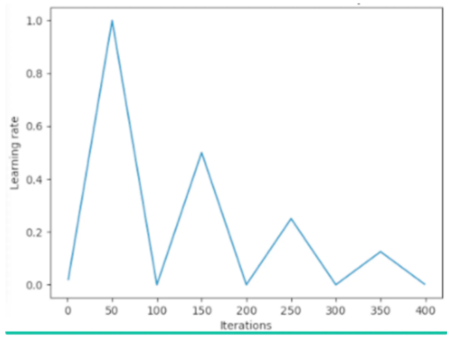


https://ruder.io/optimizing-gradient-descent/

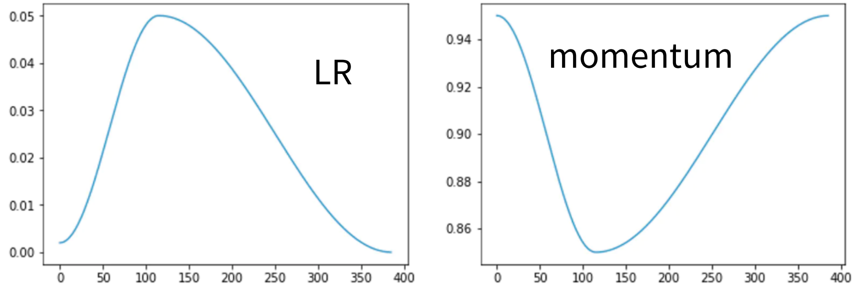
https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c

Schedulers

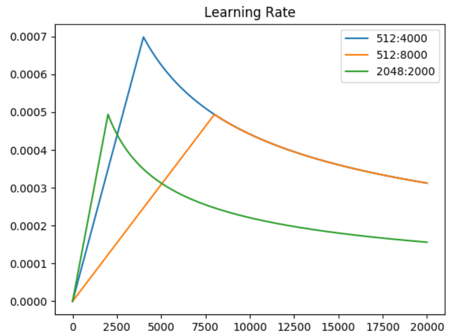
- more fiddling with LR – **warm-ups**
  - start learning slowly, then increase LR, then reduce again
  - may be repeated (**warm restarts**), with lowered maximum LR
    - allow to diverge slightly – work around local minima
- multiple options:
  - cyclical (=warm restarts) – linear, cosine annealing
  - **one cycle** – same, just don't restart
  - **Noam scheduler** – linear warm-up, decay by $\sqrt{\text{steps}}$
- combine with base SGD or Adam/Adadelta etc.
  - momentum updated inversely to LR
  - may have less effect with optimizers
    - trade-off: speed vs. sensitivity to parameter settings



cyclical scheduler (warm restarts)



one cycle with cosine annealing

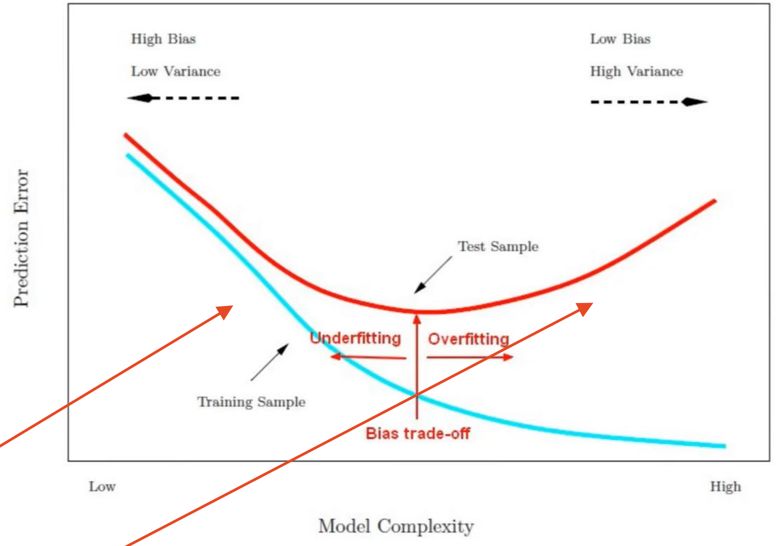


Noam scheduler with different parameters

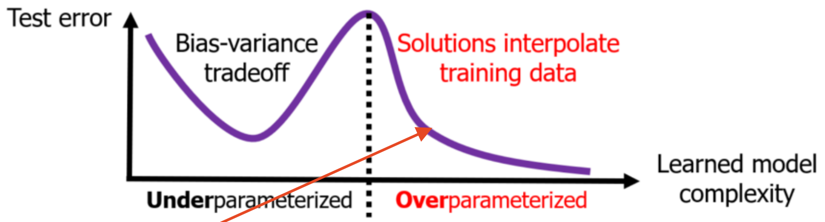
https://spell.ml/blog/lr-schedulers-and-adaptive-optimizers-YHmwMhAAACYADm6F
https://nn.labml.ai/optimizers/noam.html

When to stop training

https://www.andreaperlato.com/theorypost/bias-variance-trade-off/



- generally, when cost stops going down
  - despite all the LR fiddling
- problem: **overfitting**
  - cost low on training set, high on validation set
  - network essentially memorized the training set
  - → **check on validation set** after each epoch (pass through data)
  - stop when cost goes up on validation set
  - regularization (e.g. dropout) helps delay overfitting

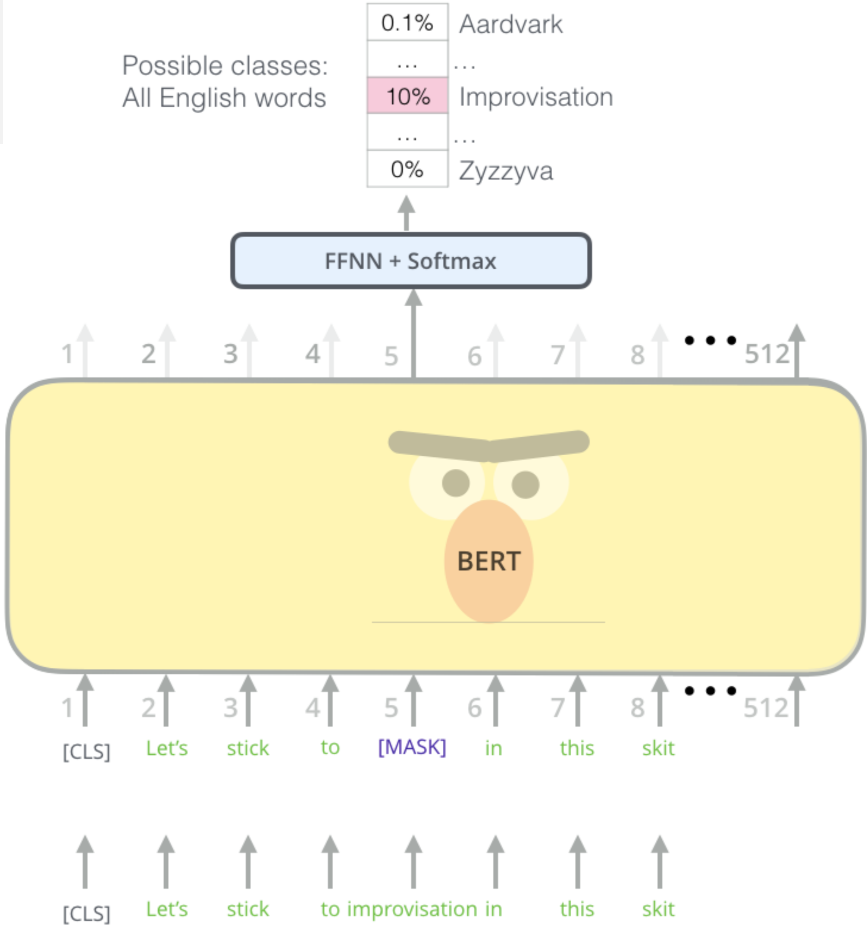


- **bias-variance** trade-off:
  - smaller models may underfit (high bias, low variance = not flexible enough)
  - larger models likely to overfit (too flexible, memorize data)
  - XXL models: overfit soo much they actually interpolate data → good (🤔 ?)
    - "grokking": model overfits → long nothing → starts generalizing

(Dar et al., 2021) https://arxiv.org/abs/2109.02355
(Power et al., 2022) http://arxiv.org/abs/2201.02177

Self-supervised training

- train supervised, but **don’t provide labels**
  - use naturally occurring labels
  - create labels automatically somehow
    - corrupt data & learn to fix them
    - learn from rule-based annotation (not ideal!)
  - use specific tasks that don’t require manual labels
- good to train on huge amounts of data
  - language modelling
    - **next-word prediction** (~ most LLMs)
    - **MLM** – masked word prediction (~ encoder LMs, e.g. BERT)
- good to **pretrain** a LM self-supervised
  before you **finetune** it fully supervised (on your own task-specific data)



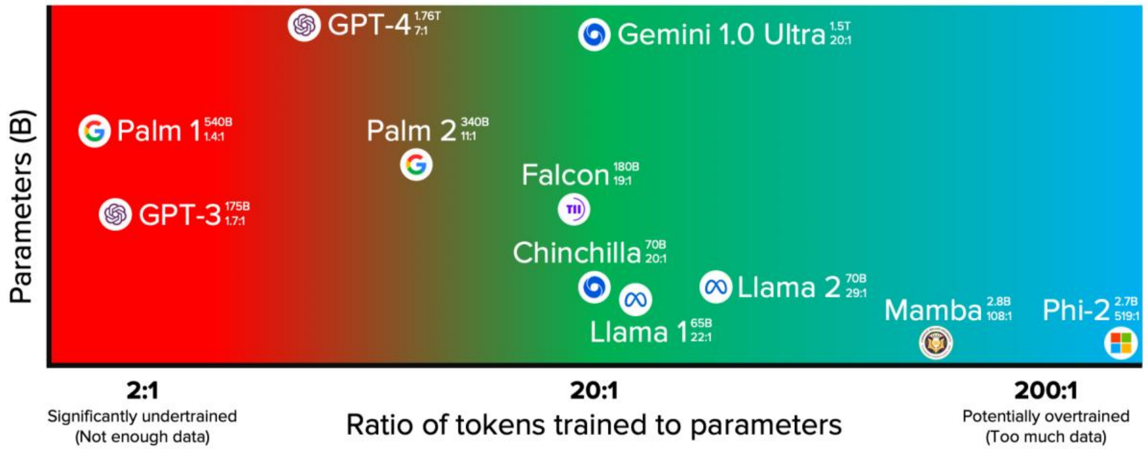
http://jalammar.github.io/illustrated-bert/

https://ai.stackexchange.com/questions/10623/what-is-self-supervised-learning-in-machine-learning

Pretraining & Finetuning: Pretrained LMs

- 2-step training:
  1. **Pretrain** a model on a huge dataset (**self-supervised**, language-based tasks)
  2. **Fine-tune** for your own task on your smaller data (**supervised**)
- ~ pretrained "contextual embeddings" ("better word2vec", typically Transformer)
- Model capability is all about the data
  - the larger model, the more you need ("Chinchilla scaling laws")
  - anyway the more, the better

https://twitter.com/Thom_Wolf/status/1766783830839406596



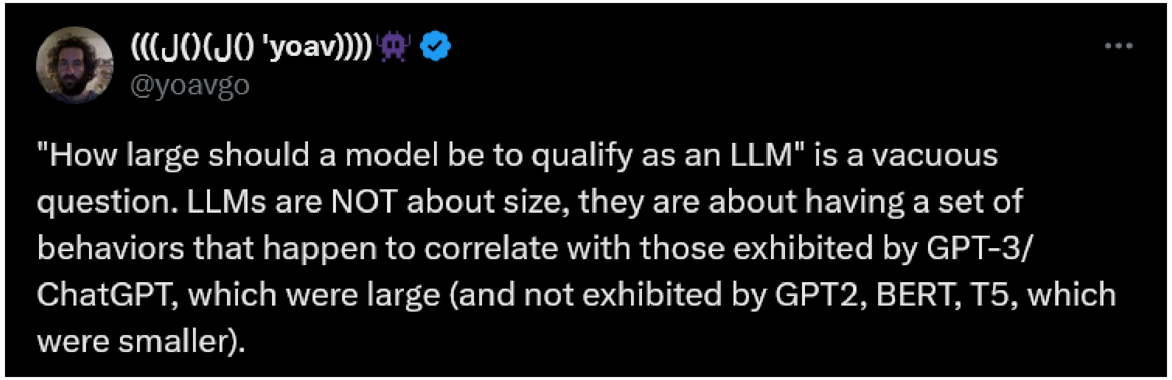
https://lifearchitect.ai/chinchilla/
https://www.harmdevries.com/post/model-size-vs-compute-overhead/

Pretrained (Large) Language Models (PLMs/LLMs)

(Zhao et al., 2023)
http://arxiv.org/abs/2303.18223

- **BERT**/**RoBERTa**/**ModernBERT**: Transformer encoder
  - trained by masked language modeling, good for classification
- **GPT-2,** most LLMs(**GPT-3/4**,**Llama**,**Mistral**,**Gemma**, **Phi**, **Qwen**…): decoder
  - trained by next-word prediction (=language modeling), good for generation / prompting
- **BART, T5** – encoder-decoder (many training tasks, good for generation)
- multilingual: **XLM-RoBERTa**, **mBART**, **mT5**, **Aya**
- many models released plug-and-play
  - **!!** others (GPT-3/3.5/4, Claude… closed & API-only)
- PLM vs. LLM distinction a bit vague
  - generally >1B, but more on behavior
  - PLMs: ready to finetune
  - LLMs: ready to prompt (→→)

https://huggingface.co/
https://ollama.com/

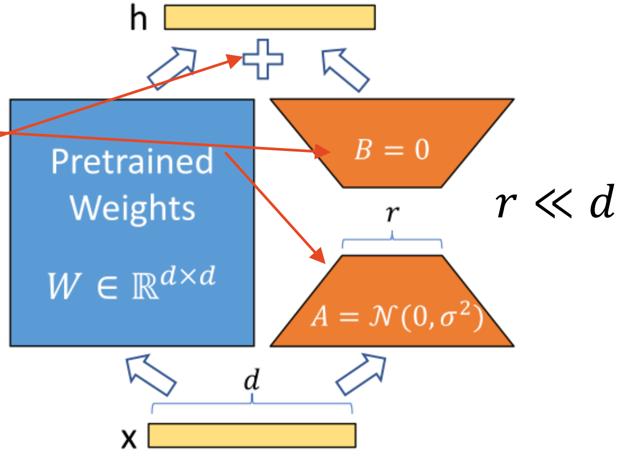


(controversial! see discussion 🍿) https://x.com/yoavgo/status/1828383882317549765

Parameter-efficient Finetuning

(Lialin et al., 2023) http://arxiv.org/abs/2303.15647
(Sabry & Belz, 2023) http://arxiv.org/abs/2304.12410

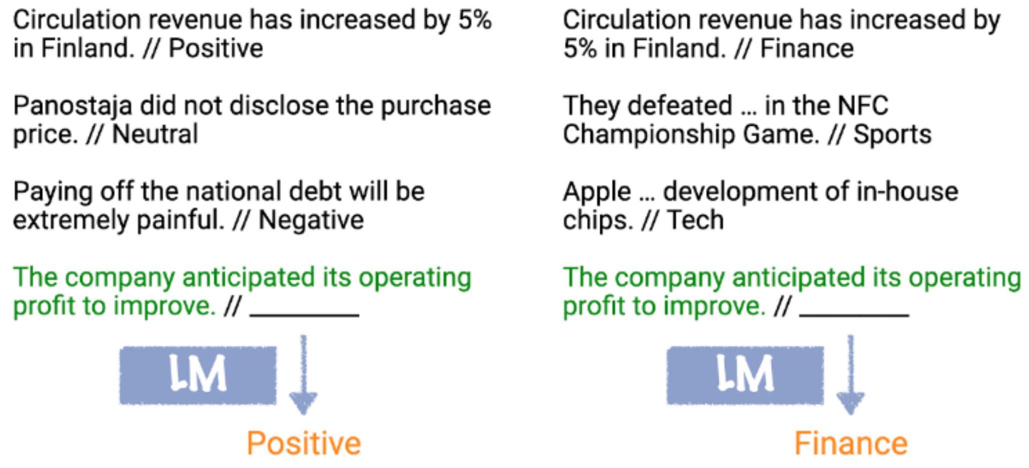
- Finetuning large models: don't update all parameters
  - less memory-hungry (fewer gradients/momentums etc.)
  - trains faster
  - less prone to overfitting (~ regularization)
- Add few parameters & only update these
  - **Adapters** – small feed-forward networks after/on top of each layer
  - **Soft prompts** – tune a few special embeddings & use them on input
  - **LoRA** (low-rank adaptation):
    - 2 decomposition matrixes A, B (parallel to each layer)
    - update = multiplication AB
    - $2 \times r \times d$ is much smaller than full weights (d^2)
    - update is added to original weights on the fly
  - **QLoRA** – LoRA + quantized 4/8-bit computation
    - to fit large models onto a small GPU



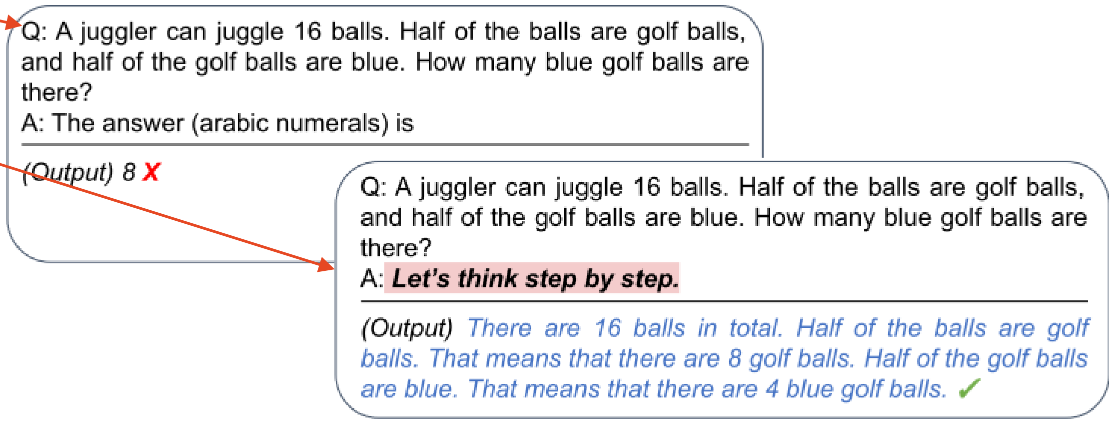
(Houlsby et al., 2019) http://proceedings.mlr.press/v97/houlsby19a.html
(Lester et al., 2021) https://aclanthology.org/2021.emnlp-main.243
(Hu et al., 2021) http://arxiv.org/abs/2106.09685
(Dettmers et al., 2023) http://arxiv.org/abs/2305.14314

LLMs: Prompting = In-context Learning

- No model finetuning, just show a few examples in the input (=prompt)
- pretrained LMs can do various tasks, given the right prompt
  - they've seen many tasks in training data
  - only works with the larger LMs (>1B)
- adjusting prompts often helps
  - "**prompt engineering**"
  - **zero-shot** (no examples) vs. **few-shot**
  - **chain-of-thought** prompting: "let's think step by step"
  - adding / rephrasing **instructions** (see → →)



http://ai.stanford.edu/blog/understanding-incontext/



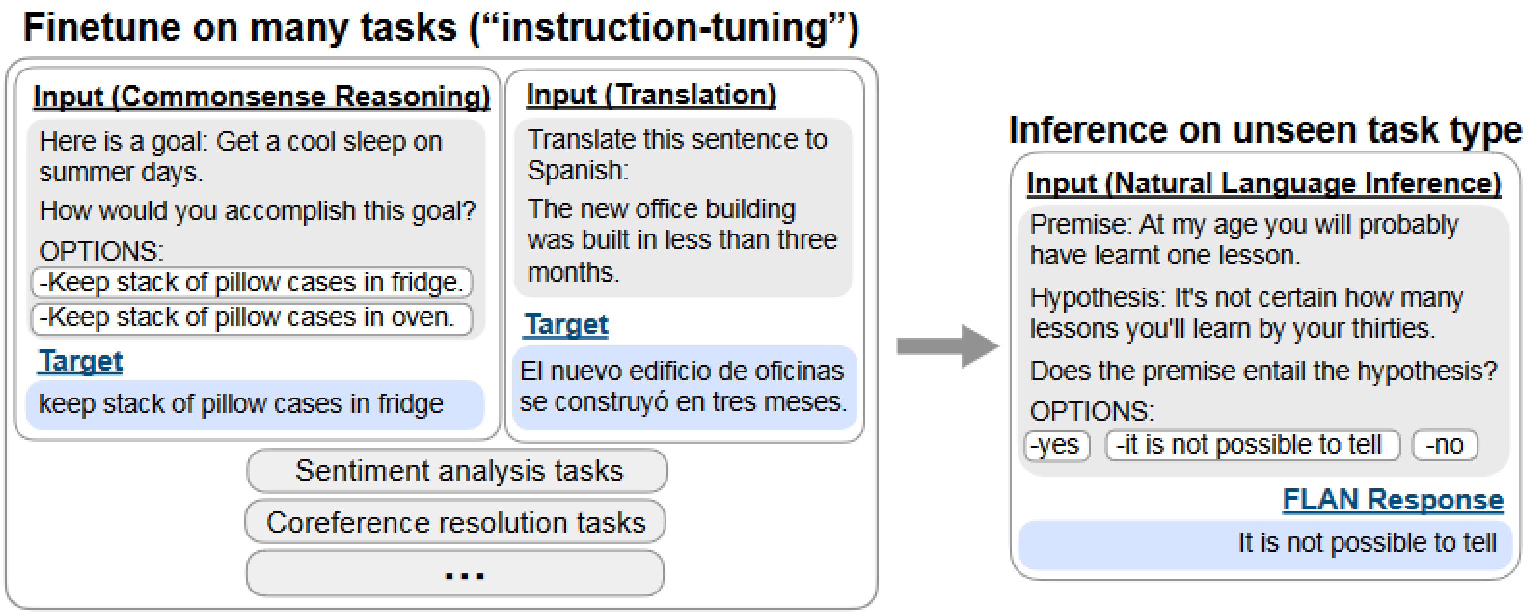
https://lilianweng.github.io/posts/2023-03-15-prompt-engineering/

(Liu et al., 2023) https://arxiv.org/abs/2107.13586

Instruction Tuning

(Wei et al., 2022) https://arxiv.org/abs/2109.01652

- Finetune for use with prompting
  - "in-domain" for what it's used later
- Use **instructions** (task description) **+ solution** in prompts
  - Many different tasks, specific datasets available
- Some LLMs released as base ("foundation") & instruction-tuned versions

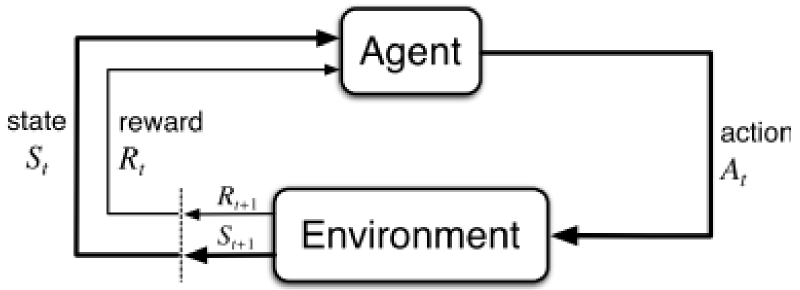


https://nlpnewsletter.substack.com/p/instruction-tuning-vol-1

Reinforcement Learning

- Learning from **weaker supervision**
  - only get feedback once in a while, not for every output
  - good for globally optimizing sequence generation
    - you know if the whole sequence is good
    - you don't know if step X is good
  - sequence ~ whole generated text
- Framing the problem as **states & actions & rewards**
  - "robot moving in space", but works for text generation too
  - state = generation so far (prefix)
  - action = one generation output (subword)
  - defining rewards is an issue (→→)
- Training: **maximizing long-term reward**
  - optimizing policy = way of choosing actions, i.e. predicting tokens

your model



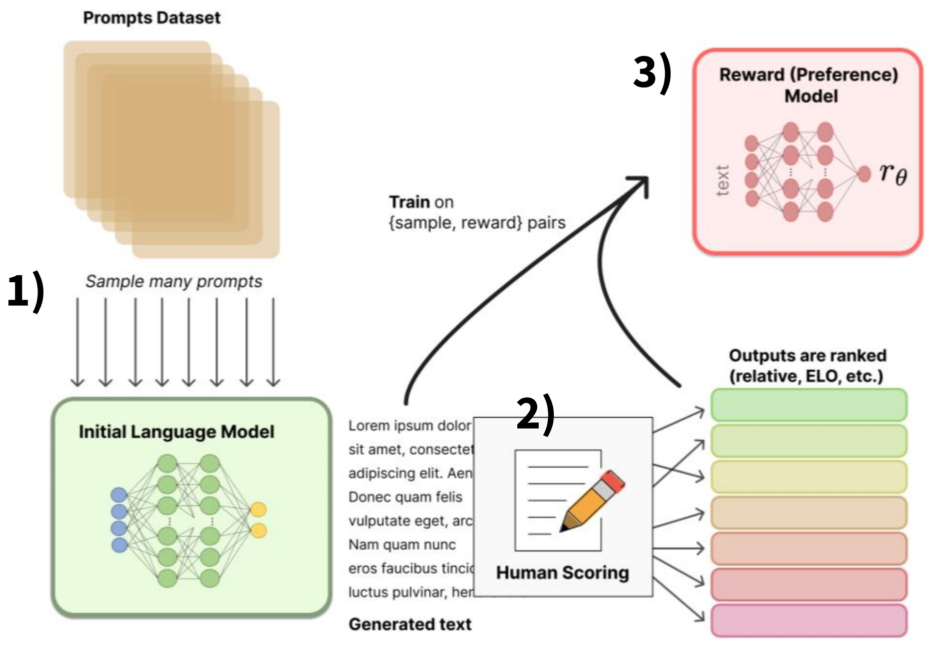
(Sutton & Barto, 2018)

some definition of rewards

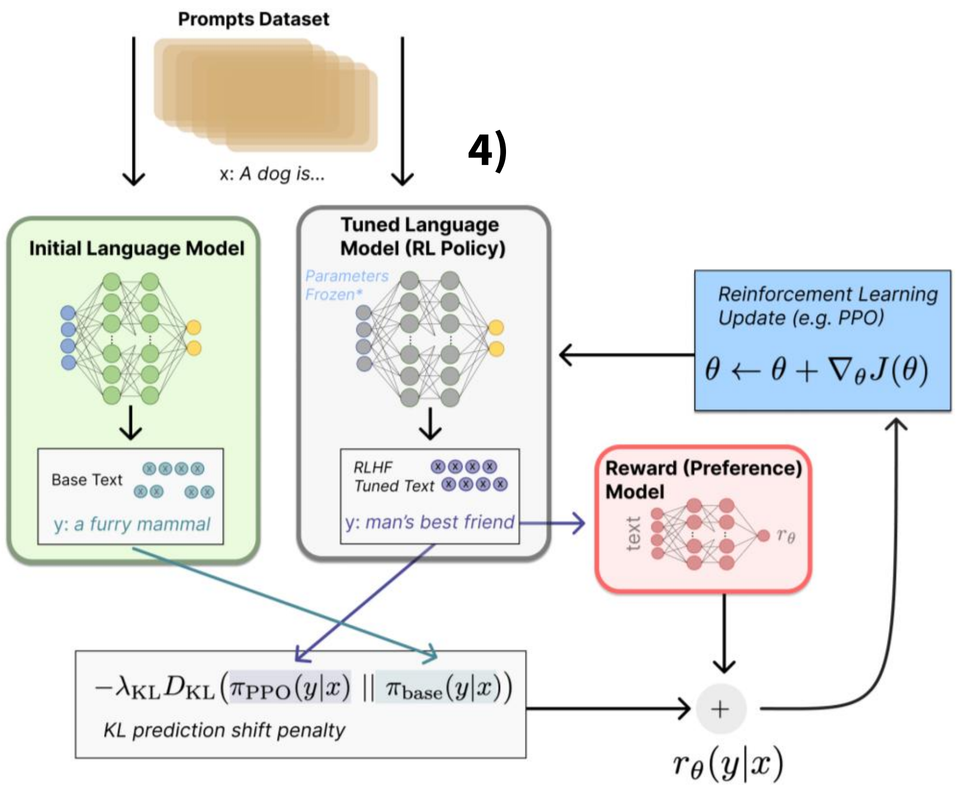
RL from Human/AI Feedback (RLHF/RLAIF)

(Ouyang et al., 2022)
http://arxiv.org/abs/2203.02155
https://openai.com/blog/chatgpt

- RL improvements on top of instruction tuning (~InstructGPT/ChatGPT):
  1) generate lots of outputs for instructions
  2) have humans rate them (**RLAIF variant**: replace humans with an off-the-shelf LLM)
  3) learn a reward model (some kind of other LM: instruction + solution → score)
  4) use rating model's score as reward in RL
  - main point: **reward is global** (not token-by-token)



https://huggingface.co/blog/rlhf



Direct Preference Optimization

(Rafailov et al., 2023) http://arxiv.org/abs/2305.18290

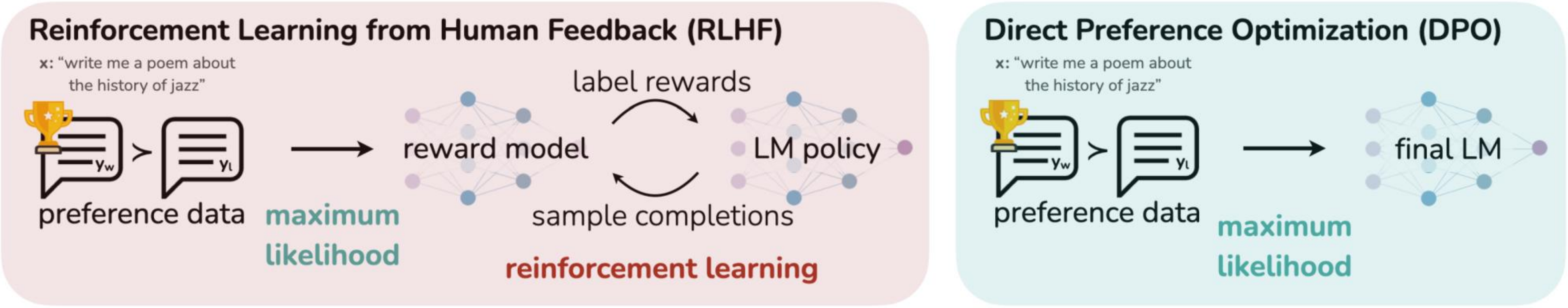
- Trying to do the same thing, but without RL, with supervised learning
- Special loss function to check pairwise text preference
  - increases probability of preferred response
  - includes weighting w.r.t. reference model

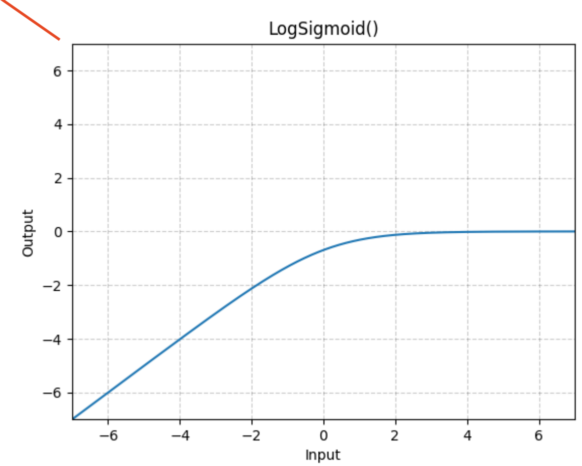
y_l dispreferred

y_w preferred

$$L_{DPO}(\pi_\theta; \pi_{ref}) = -\mathbb{E}_{(x,y_w,y_l)\sim D}\left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right]$$

optimized model

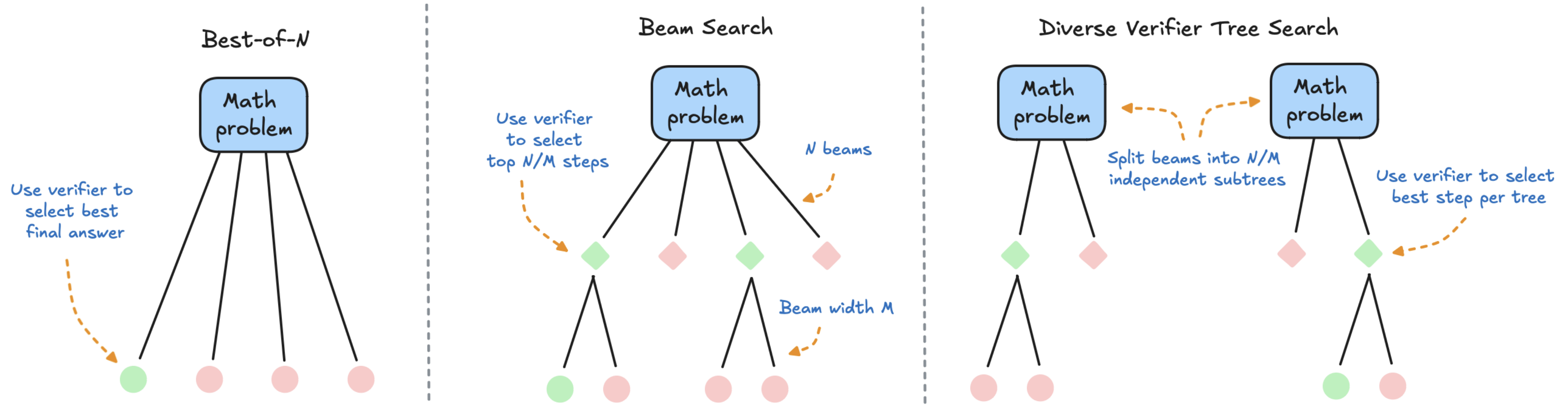




Scaling Test-time Compute – Reasoning Models

https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute
https://timkellogg.me/blog/2025/01/25/r1
(Muennighoff et al., 2025) http://arxiv.org/abs/2501.19393

- Glorified **chain-of-thought**
  - make chains very long
  - train models with intermediate rewards (process reward models)
- The longer you compute, the better
  - can be tree search (over intermediate steps, with backtrack), but linear seems OK
  - budget-forcing: inserting "Wait" / force-terminating
- RL again (GRPO: sample a lot, baseline = average, upvote better-than-average)



Synthetic Data

- Generate stuff via base model, train on the result
  - like what we did with RLHF/DPO, but for standard training – earlier & more
- Useful for
  - detailed annotation (like process rewards)
  - cleaner data
  - generally more data
  - better-aligned data (rewrite as problem-solution pairs, flip problem direction…)
  - target modality data (text → audio)
- Needs careful filtering
  - iterative refinement – model evaluates itself
  - synthetic code: validate via execution

(Abdin et al., 2024) http://arxiv.org/abs/2412.08905
(Defossez et al., 2024) https://arxiv.org/abs/2410.00037

Thanks

Contacts:

Ondřej Dušek
odusek@ufal.mff.cuni.cz
https://tuetschek.github.io
@tuetschek

Supported by: European Research Council (ERC StG No. 101039303 NG-NLG)
Using LINDAT/CLARIAH-CZ Research Infrastructure resources (Czech Ministry of Education, Youth and Sports project No. LM2018101).