FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

NI-NLM – Lecture 3

Training language models

Zdeněk Kasner

3 Mar 2026

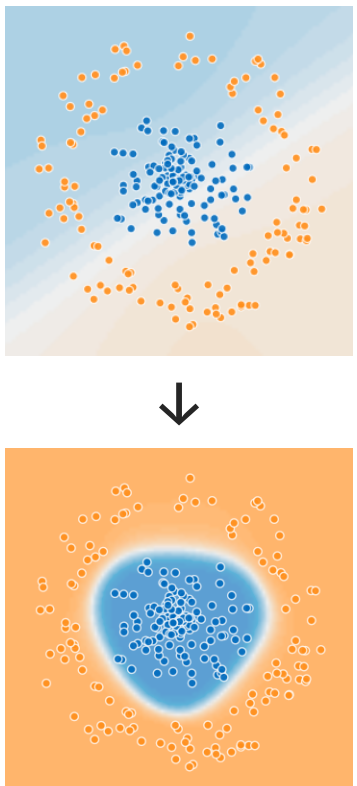
Training neural networks

source: https://playground.tensorflow.org

👨‍🍳 Recipe for supervised training of NNs

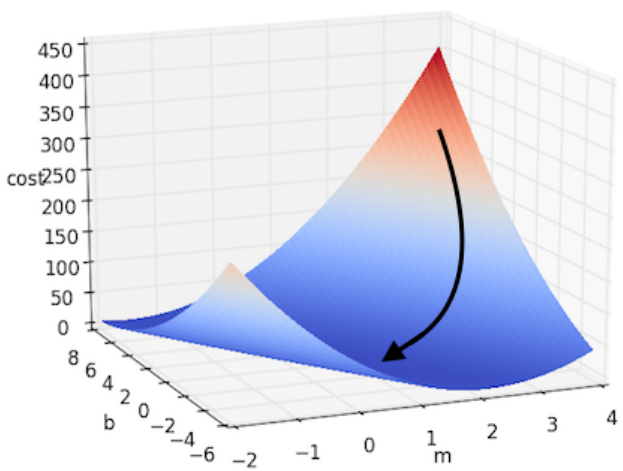
Repeat for each training example (x, y):

1. Let the model **predict** $\mathcal{M}(x) = \hat{y}$.
2. Compute a **loss function** $\mathcal{L}(\hat{y}, y)$: how wrong the model prediction $\hat{y}$ is.
3. Compute **gradients**: how each parameter of the model $\mathcal{M}$ contributed to the loss $\mathcal{L}$.
4. Apply **backpropagation**: update the model $\mathcal{M}$ parameters in the direction that reduces the loss $\mathcal{L}$.

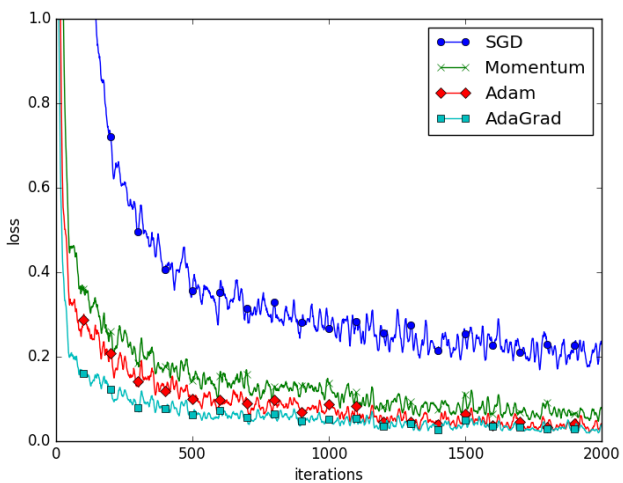


How does training a neural network look like?

source: https://playground.tensorflow.org



source: https://ml4a.github.io



source: https://kaeken.hatenablog.com/

How does training a Transformer look like?

source: Karpathy's microgpt

```
# Repeat in sequence
num_steps = 1000 # number of training steps
for step in range(num_steps):

    # Take single document, tokenize it, surround it with BOS special token on both sides
    doc = docs[step % len(docs)]
    tokens = [BOS] + [uchars.index(ch) for ch in doc] + [BOS]
    n = min(block_size, len(tokens) - 1)

    # Forward the token sequence through the model, building up the computation graph all the way to the loss
    keys, values = [[] for _ in range(n_layer)], [[] for _ in range(n_layer)]
    losses = []
    for pos_id in range(n):
        token_id, target_id = tokens[pos_id], tokens[pos_id + 1]
        logits = gpt(token_id, pos_id, keys, values)
        probs = softmax(logits)
        loss_t = -probs[target_id].log()
        losses.append(loss_t)
    loss = (1 / n) * sum(losses) # final average loss over the document sequence. May yours be low.
```

```
    # Backward the loss, calculating the gradients with respect to all model parameters
    loss.backward()

    # Adam optimizer update: update the model parameters based on the corresponding gradients
    lr_t = learning_rate * (1 - step / num_steps) # linear learning rate decay
    for i, p in enumerate(params):
        m[i] = beta1 * m[i] + (1 - beta1) * p.grad
        v[i] = beta2 * v[i] + (1 - beta2) * p.grad ** 2
        m_hat = m[i] / (1 - beta1 ** (step + 1))
        v_hat = v[i] / (1 - beta2 ** (step + 1))
        p.data -= lr_t * m_hat / (v_hat ** 0.5 + eps_adam)
        p.grad = 0

    print(f"step {step+1:4d} / {num_steps:4d} | loss {loss.data:.4f}", end='\r')
```

Loss function

A **loss function** measures **how far off** the model's prediction is from the correct answer.

How do we measure that? Depends on our problem.

> **Question**
>
> Assume your model predicts a **real number**. How would you measure the error with respect to the ground truth value?

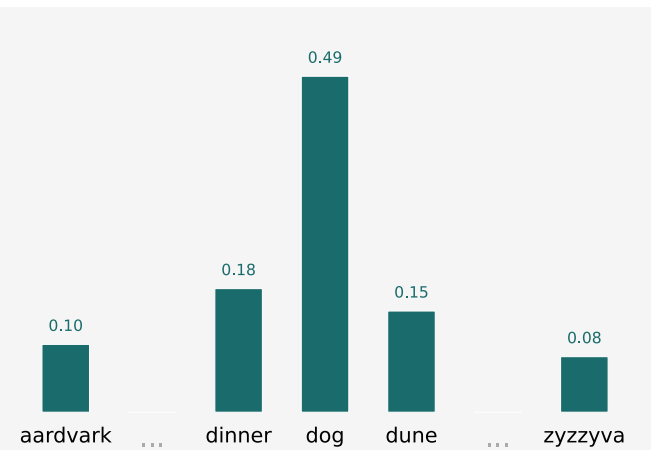
- `(true - predicted)`: bad idea, the errors may cancel out
- `abs(true - predicted)`: ok, but non-differentiable at 0 → training issues
- `(true - predicted)^2`: works well → smooth, penalizes outliers

Loss function: language modeling

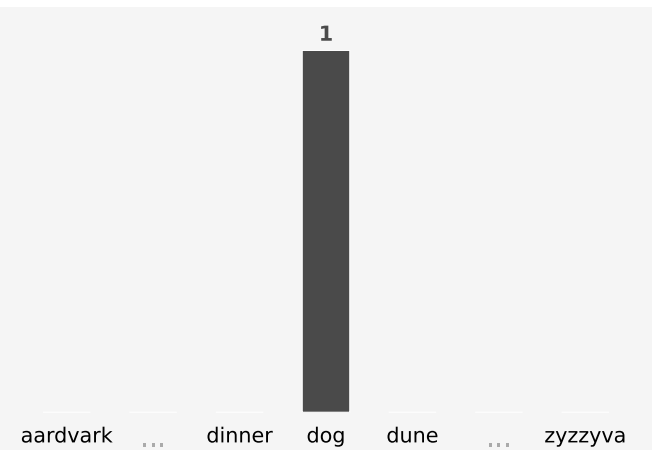
Question

Assume your model predicts a **probability distribution for the next word**. How would you measure the error with respect to the ground truth value?

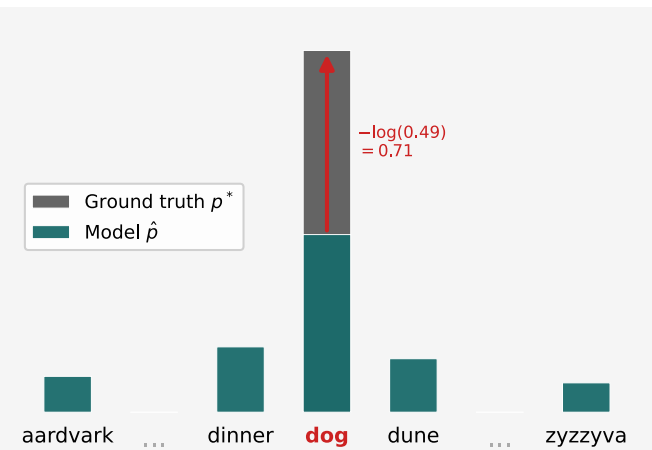
Model prediction



Ground truth



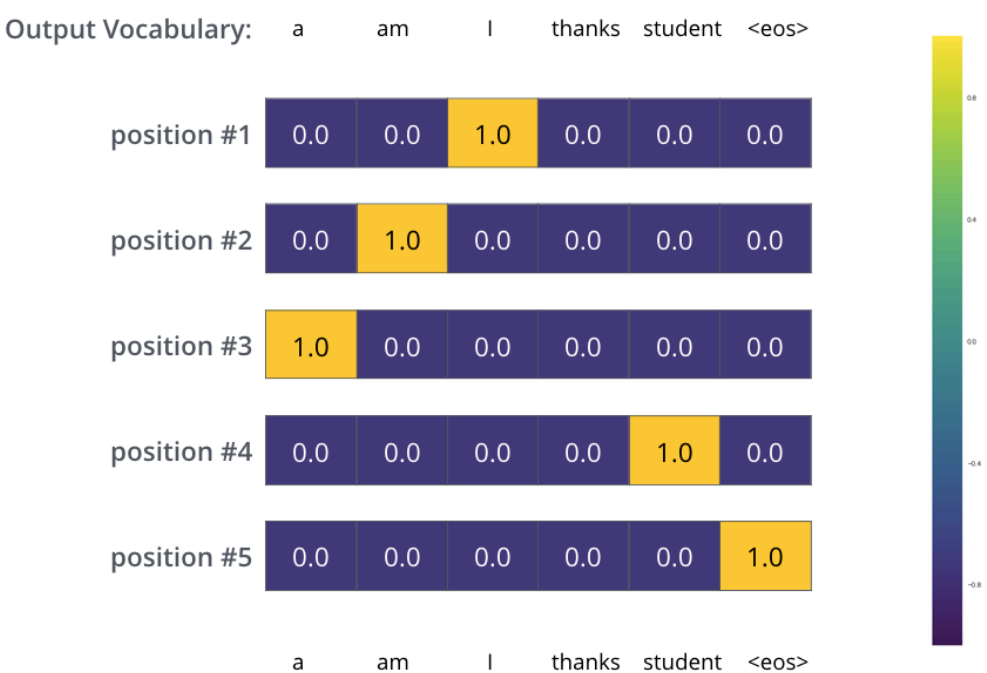
Model error



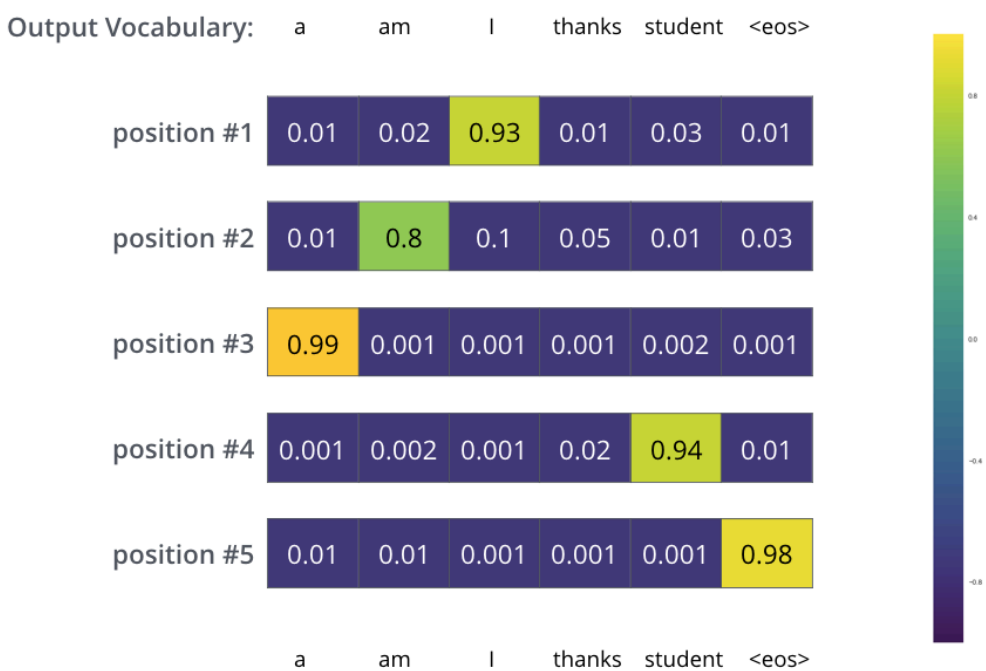
Loss function: language modeling

source: The Illustrated Transformer

What we train for:



What we hope to get:



Cross-entropy loss

The loss function used for the next word prediction task is called **cross-entropy**:

$$\mathcal{L} = -\sum_{c=1}^{C} y_c \cdot \log(\hat{y}_c)$$

where y_c is the true label (one-hot) and $\hat{y}_c$ is the predicted probability.

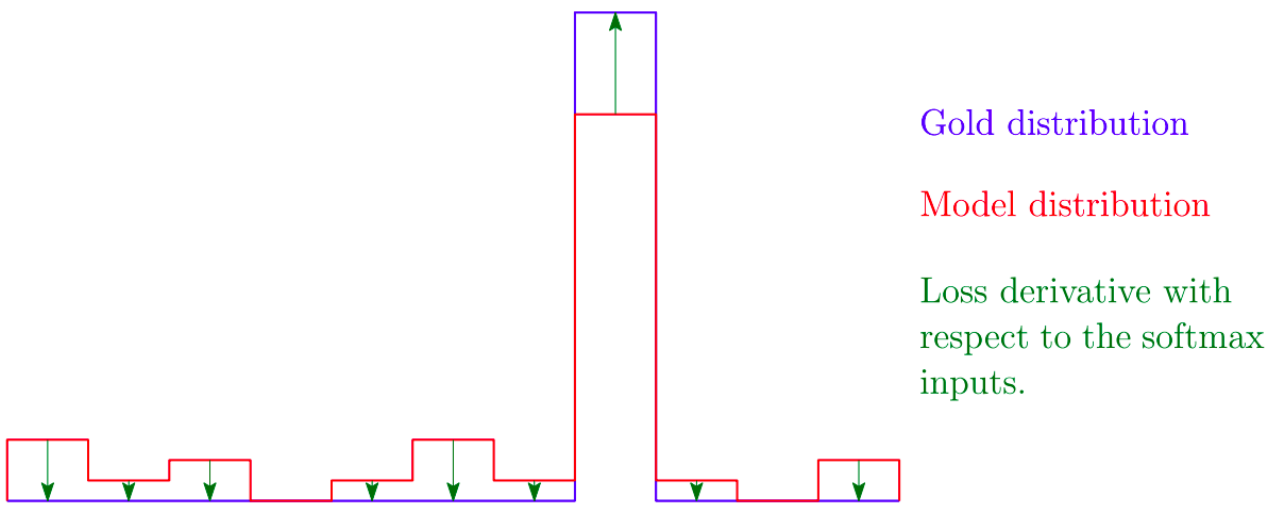
Intuition about cross-entropy for language modeling

- The model is supposed to predict the target token with **100% probability** and other tokens with 0% probability → any difference counts as an error.
- The loss is the **negative log of the predicted target token probability** (why?)

Loss gradient – softmax input

source: NPFL138

The gradient with respect to the **softmax input**:

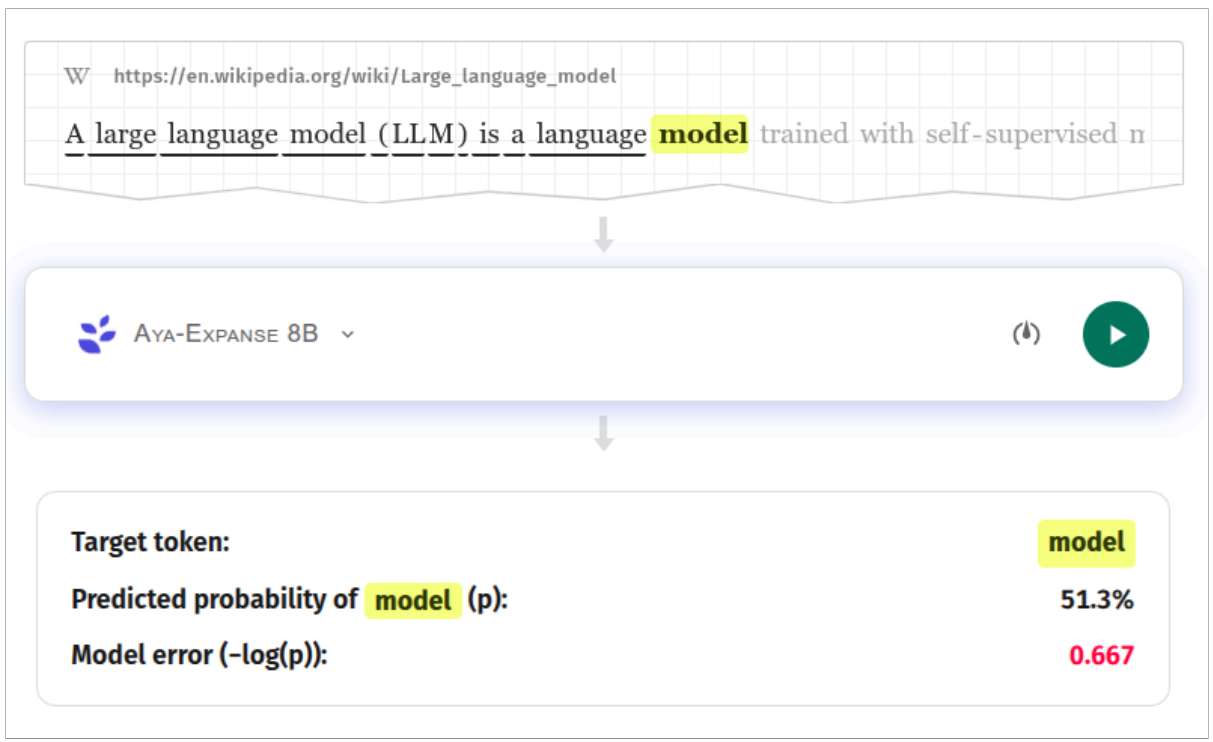


→ contributions towards the ground truth token probability are **promoted**, contributions towards other tokens probabilities are **discouraged**.

Cross-entropy in language models

Let's see it animated:

https://animatedllm.github.io/pretraining-simple



Descending the loss landscape

source: Hackernoon

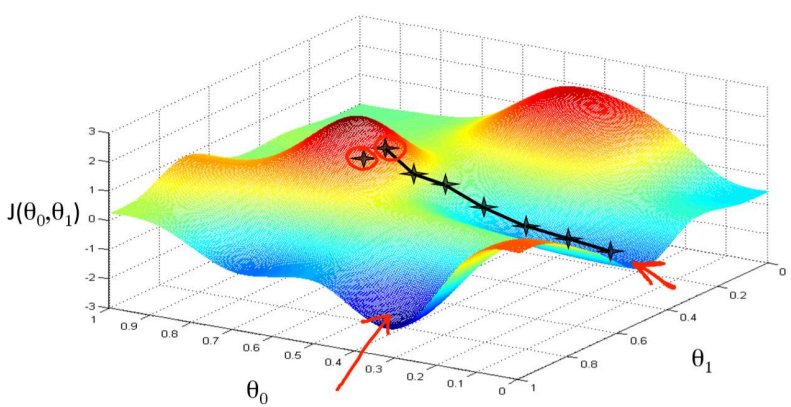
Our goal is to **minimize the loss** by iteratively **updating the model parameters** θ in the direction that reduces the loss $\mathcal{L}$:

Gradient descent as a math formula

$$\theta \leftarrow \theta - \eta \nabla \mathcal{L}(\theta)$$

This simply says: we compute the steepest ascent direction (∇) and go the opposite way.

- ∇ is the **gradient** of the loss w.r.t. each parameter.
- η is the **learning rate**.

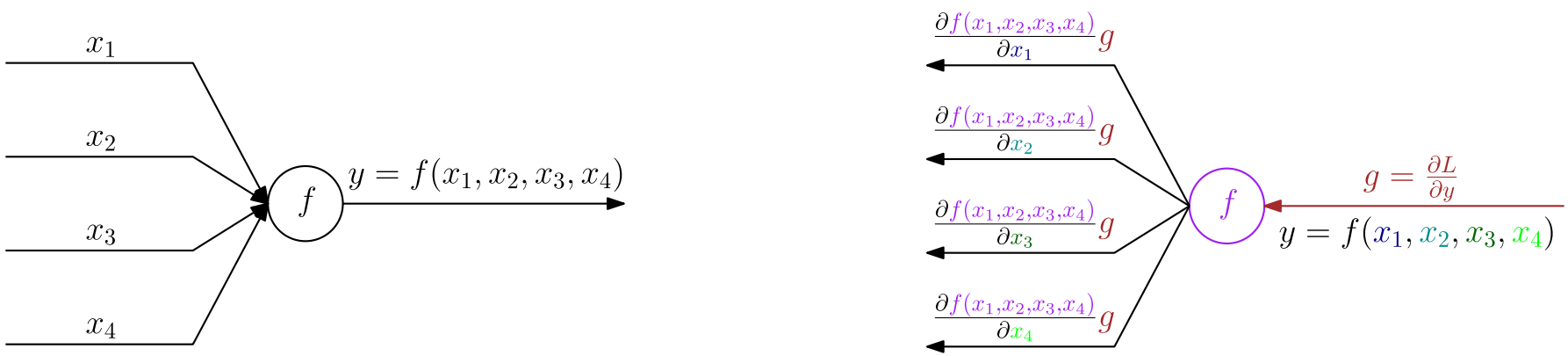


Computing gradient

source: NPFL138

Question

We ended the forward pass by computing the loss. How do we get the gradient?



We start by computing $\frac{\partial \mathcal{L}}{\partial y}$ → **the derivative of the loss with respect to the model's output probabilities** (the softmax outputs).

Computing gradient

Since our loss is computed as:

$$\mathcal{L} = -\log y_{\text{correct}},$$

then:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_c} = \begin{cases} -\frac{1}{\hat{y}_c} & \text{for the correct class} \\ 0 & \text{for all other classes} \end{cases}$$

Gradient vs. derivative

Gradient (∇) is how a (partial) **derivative** ∂ is called for multivariate functions (= functions operating on multi-dimensional inputs).

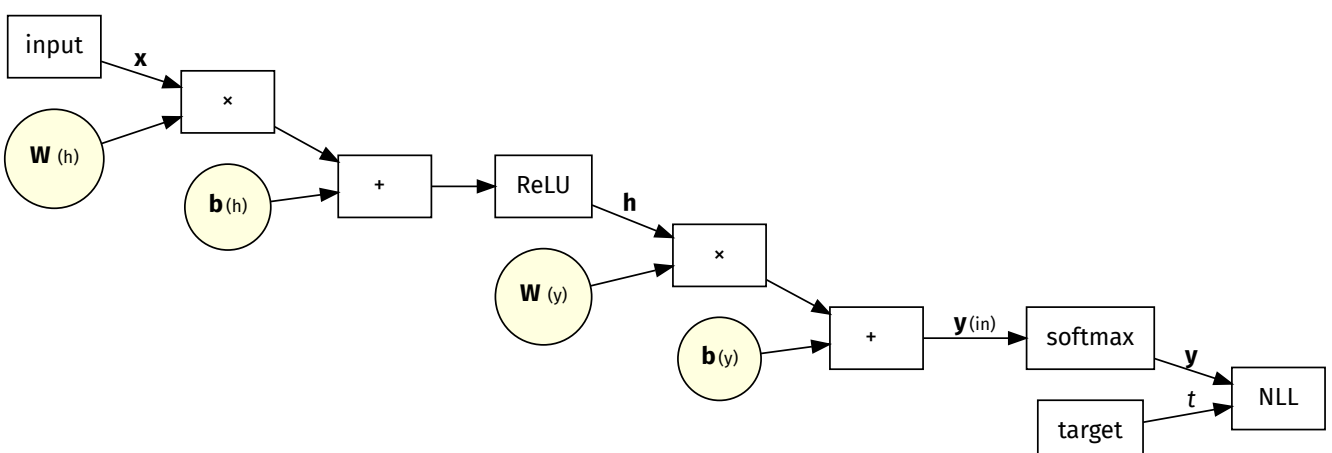
Backpropagation

source: NPFL129

Backpropagation = repeated application of the **chain rule** of derivatives for $f(g(x))$:

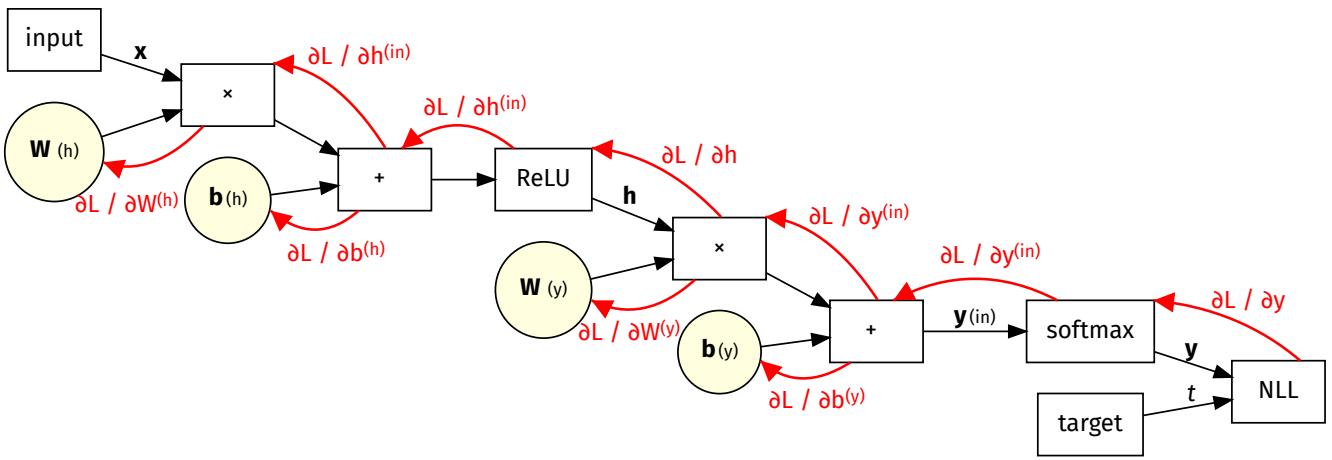
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x}$$

How to apply it? First, build a **computation graph** (example for a simple FFNN):



Backpropagation

...then apply the **chain rule** in the backward direction:



Computing the gradient in practice

Modern deep learning frameworks (PyTorch, TensorFlow) have built-in **autograd** functionality that automatically computes gradients for us.

Stochastic gradient descent

Computing the gradient over the **entire dataset** at once is ~~expensive~~ impossible.

Idea

Instead of using all training examples, we iteratively compute the gradient on **small random subsets** of the training examples.

- **Stochastic gradient descent (SGD)**: our subset is of size 1 (=a single example).
- **Mini-batch SGD**: we take a batch of examples (e.g. 32 examples).

Question

Is it useful to introduce some stochasticity into the training process?

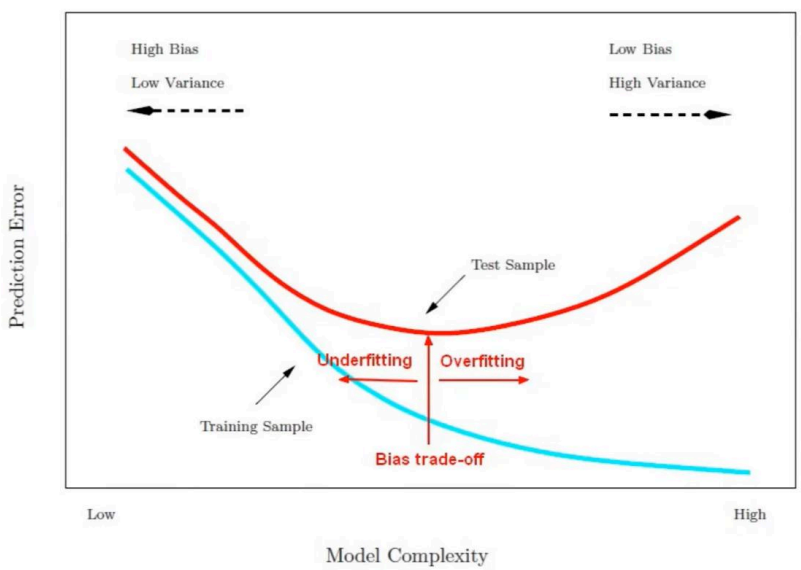
Learning rate

source: CS231n

The **learning rate** η controls how big the parameter update steps are.

Intuition about learning rate

- **Too high** → may overshoot the minimum, training diverges.
- **Too low** → converges very slowly, may get stuck in a bad local minimum.
- It is usually a small multiplier: 0.01, 0.001, ...



Into the depths of deep learning

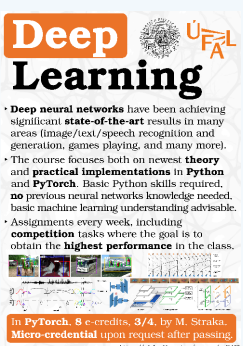
There are other concepts for NN training that we do not cover here, such as:

- **Weight initialization**: how do we initialize the model weights before training?
- **Regularization**: how do we prevent the model from overfitting to training data?
- **Optimizers, momentum**: is there a more efficient approach than SGD?

If you want to learn more...

See the MFF UK **Deep Learning course** (lectures 1-3, lecture recordings available):

https://ufal.mff.cuni.cz/courses/npfl138/2526-summer



Pretraining the Transformer

Supervised training?

source: Vaswani et al. (2017)

The original Transformer was trained **fully supervised on machine translation**:

- English → German (WMT 2014, 4.5M sentence pairs)
- English → French (WMT 2014, 36M sentence pairs)

> **Question**
>
> Where can we get that much data for other tasks?

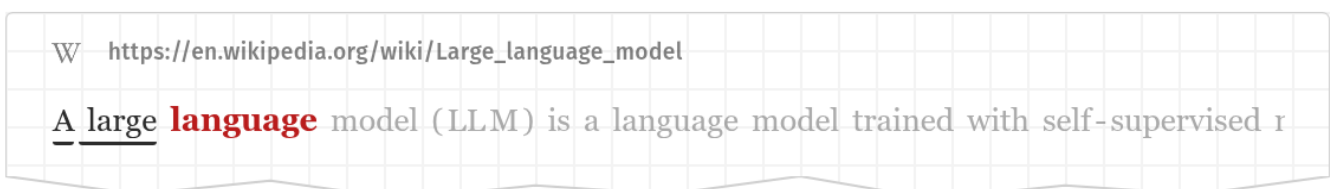
We do **not** have that much data for other tasks 😢

But much of what the model was learning during the training was the **structure of the language** itself...

Self-supervised pretraining

Idea

If we train the model for predicting the words in a text, **any text is training data**!



- The training regime where the labels come from the input itself is called **self-supervised learning**.
- We can use it for learning general language representations

→ …maybe we will then need less data for the specific tasks?

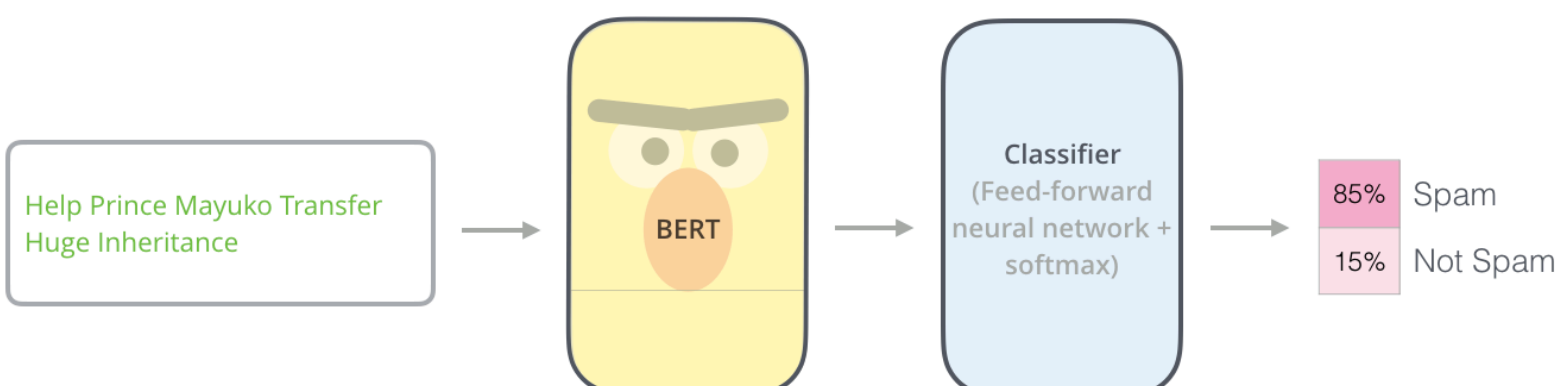
Pretraining a Transformer encoder

source: Illustrated BERT

Idea

We know the text representation is built in the Transformer **encoder**. Can we start pretraning the encoder alone?

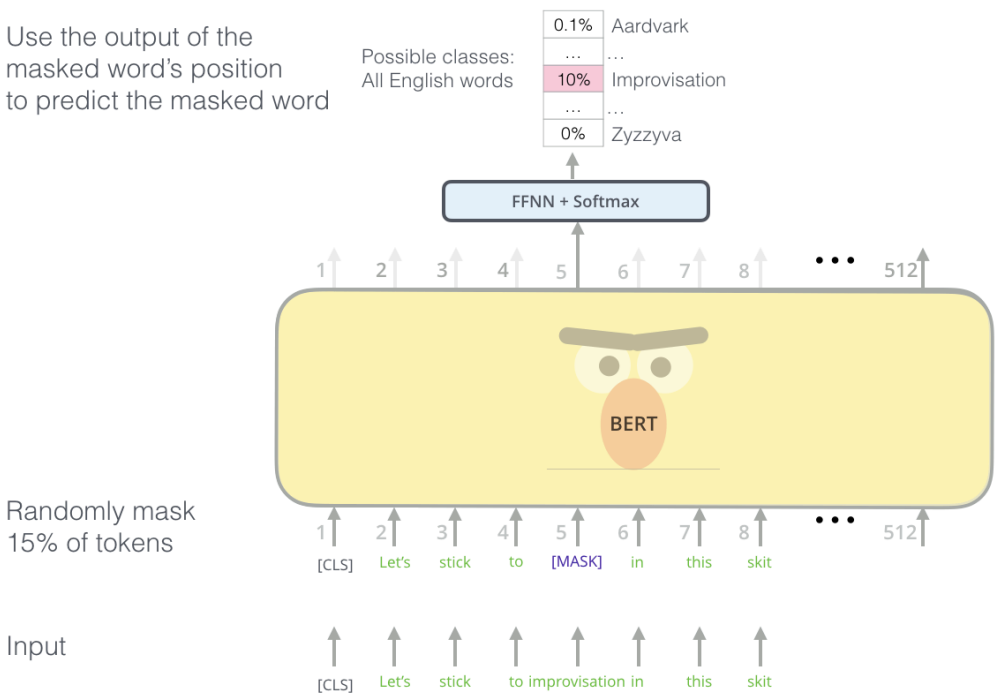
Why encoder alone? We can put classifier on top of the encoded representations to perform classification tasks:



Pretraining a Transformer encoder

Goal: Get a rich representation of the input → one *contextual* embedding per token.

Each embedding should incorporate the full sentence context (both **left and right**).



Masked language modeling

Idea

Let's mask some portion (15%? or 30%?) of words and let the model predict it.

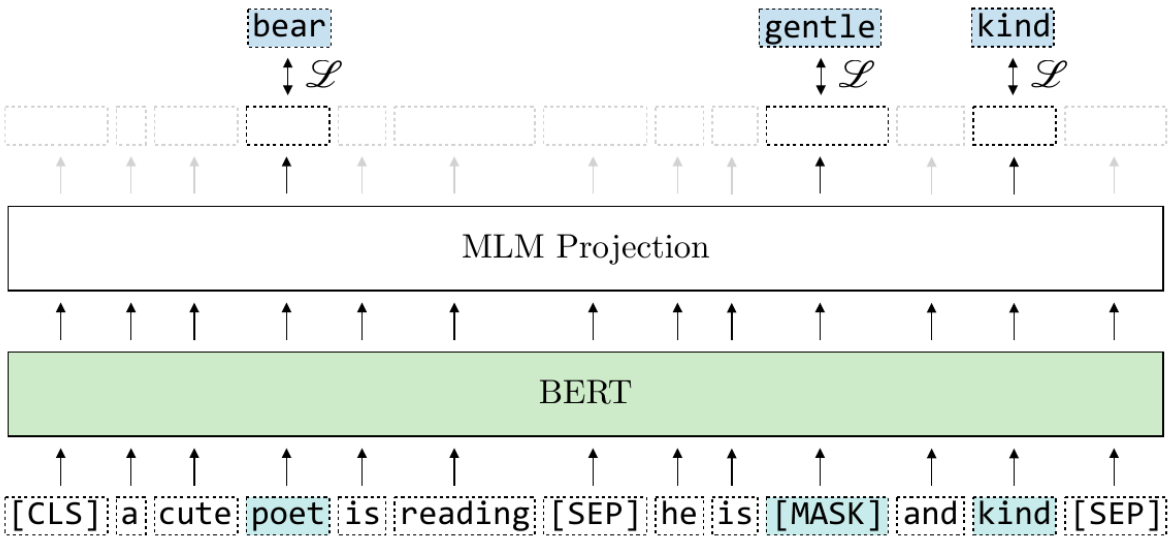
The masked language modeling (MLM) objective:

$$\mathcal{L}_{\text{MLM}} = -\sum_{t \in \mathcal{M}} \log P(x_t \mid \boldsymbol{x}_{\setminus \mathcal{M}}),$$

where $\mathcal{M}$ is the set of masked positions and $\boldsymbol{x}_{\setminus \mathcal{M}}$ is the input sequence with masked tokens replaced by a special [MASK] token.

Masked language modeling

source: Super Study Guide: Transformers & Large Language Models



Info

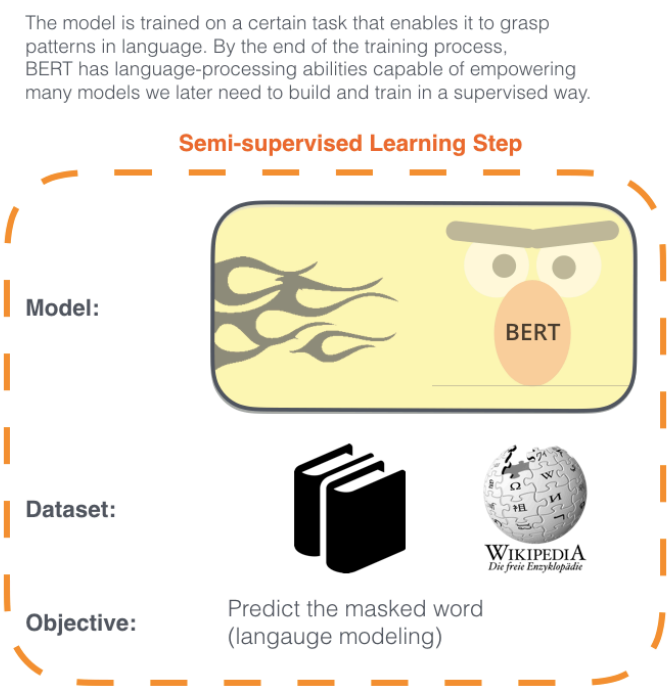
The original BERT model also used other transformations (replacing tokens, next sentence prediction), but these were dropped in follow-up models.

Pretraining a Transformer encoder

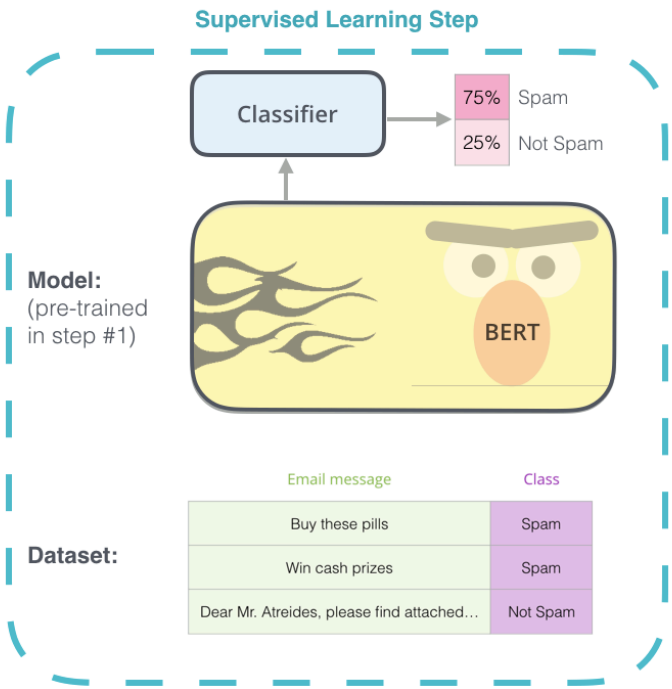
Our setup:

1 - Semi-supervised training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - Supervised training on a specific task with a labeled dataset.



BERT: Pre-trained Transformer encoder

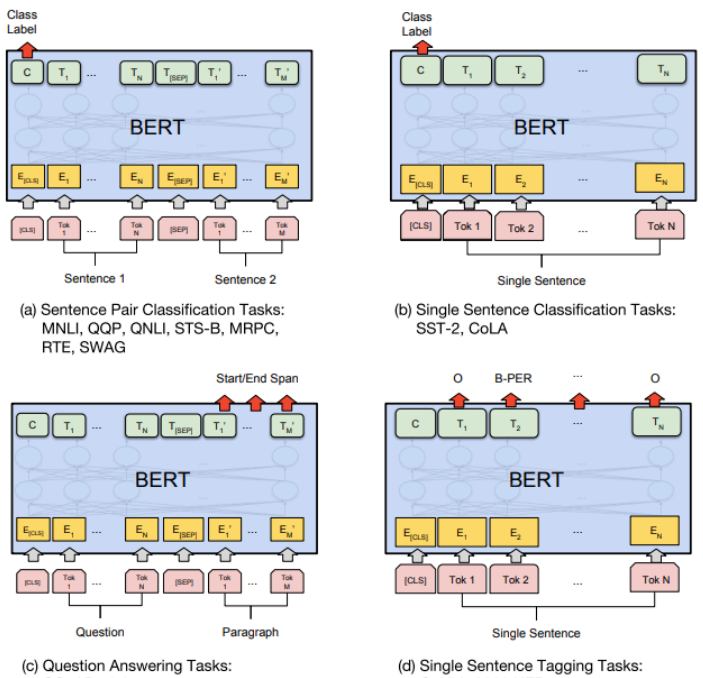
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

(Devlin et al., 2019) → breakthrough approach for many NLP tasks.

Info

The results BERT was getting were very cool – anything using BERT was suddenly **state-of-the-art**. No one was sure why.

This started a whole new research field called "BERTology".



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

The Sesame Street family

source: https://muppet.fandom.com/wiki/Sesame_Street

Deep contextualized word representations

Matthew E. Peters[†], **Mark Neumann**[†], **Mohit Iyyer**[†], **Matt Gardner**[†],
{matthewp,markn,mohiti,mattg}@allenai.org

Christopher Clark[*], **Kenton Lee**[*], **Luke Zettlemoyer**[†*]
{csquared,kentonl,lsz}@cs.washington.edu

[†]Allen Institute for Artificial Intelligence
[*]Paul G. Allen School of Computer Science & Engineering, University of Washington

Abstract

We introduce a new type of *deep contextual-*

guage model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations.

source: https://arxiv.org/pdf/1802.05365

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

source: https://arxiv.org/pdf/1810.04805

Big Bird: Transformers for Longer Sequences

Manzil Zaheer, **Guru Guruganesh,** **Avinava Dubey,**
Joshua Ainslie, **Chris Alberti,** **Santiago Ontanon,** **Philip Pham,**
Anirudh Ravula, **Qifan Wang,** **Li Yang,** **Amr Ahmed**
Google Research
{manzilz, gurug, avinavadubey}@google.com

source: https://arxiv.org/pdf/2007.14062

ERNIE: Enhanced Language Representation with Informative Entities

Zhengyan Zhang[1,2,3*], **Xu Han**[1,2,3*], **Zhiyuan Liu**[1,2,3†], **Xin Jiang**[4], **Maosong Sun**[1,2,3], **Qun Liu**[4]
[1]Department of Computer Science and Technology, Tsinghua University, Beijing, China
[2]Institute for Artificial Intelligence, Tsinghua University, Beijing, China
[3]State Key Lab on Intelligent Technology and Systems, Tsinghua University, Beijing, China
[4]Huawei Noah's Ark Lab
{zhangzhengyan14,hanxu17}@mails.tsinghua.edu.cn

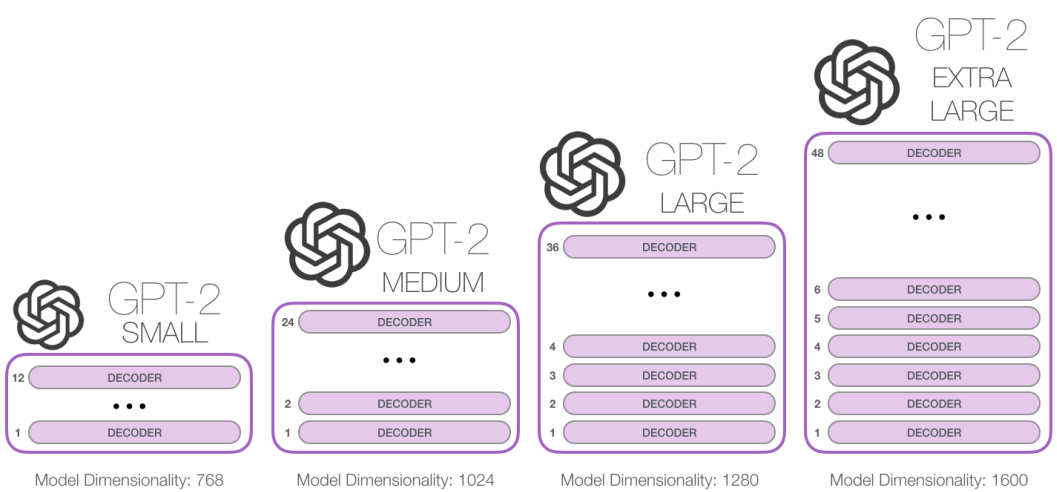
source: https://arxiv.org/pdf/1905.07129

Pretraining a Transformer decoder

source: Illustrated GPT-2

Meanwhile in OpenAI...

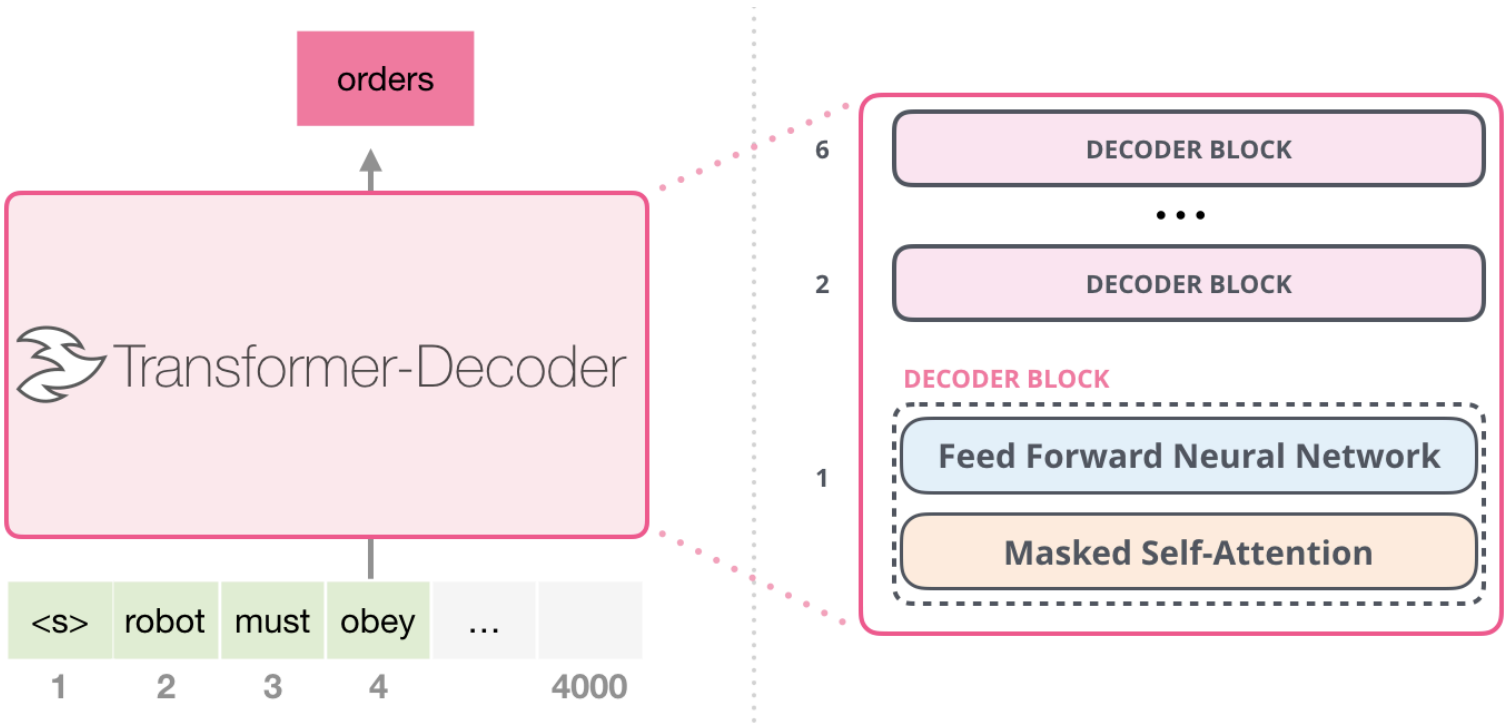
The encoder seems kind of useless. The **decoder** can represent the text on its own – and it can also **generate text**! Can we have more of that, please?



Pretraining a Transformer decoder

source: Illustrated GPT-2

Decoder-only block: only masked self-attention & FFNN, no encoder-decoder attention.



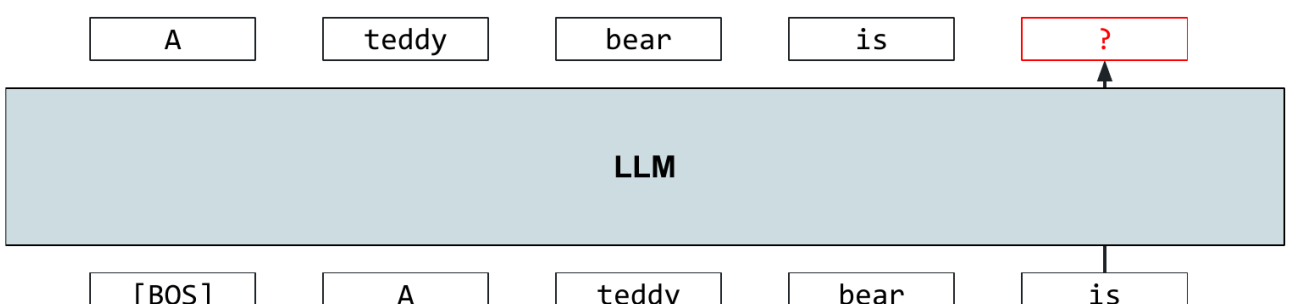
Causal language modeling

source: CME295

How to pretrain a decoder? Using the **causal language modeling** objective: what we already know as “language modeling” or “next-word prediction task”:

$$\mathcal{L} = -\sum_{t=1}^{T} \log P(x_t \mid x_1, ..., x_{t-1})$$

where x_t is the token at position t.



Attention mask

source: Super Study Guide: Transformers & Large Language Models

Recall: The attention mechanism in the decoder uses an **attention mask**.

- In practice, we apply a triangular mask during the matrix multiplications:

  → we can train all positions in parallel.

- The model learns to attend only to the left context.

$$\frac{1}{\sqrt{d_k}} QK^T + \begin{matrix} 0 & -\infty \\ 0 & 0 \end{matrix}$$

Question

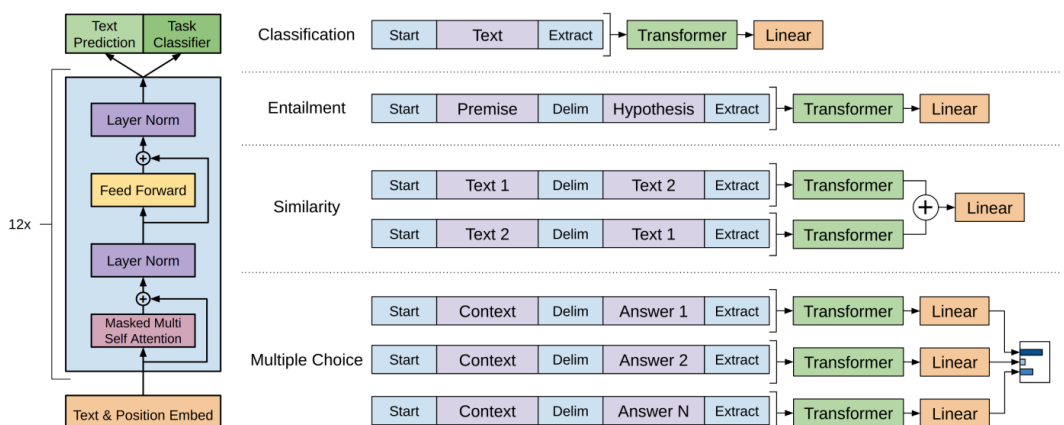
Why we do **not** need the attention mask in the encoder? And does it mean the encoder "understands" the text better?

GPT: Pre-trained Transformer decoder

source: Radford et al. 2018

The "GPT-1" paper (Improving Language Understanding by Generative Pre-Training (Radford et al., 2018))

- 12 layers, 117M params, good results on downstream tasks.
- Published (pre-print + blogpost) in June 2018 (immediately overshadowed by BERT in October 2018).



GPT: Pre-trained Transformer decoder

source: OpenAI blog

GPT-2: Language Models are Unsupervised Multitask Learners (Radford et al., 2019)

Published as a pre-print and a blogpost on the OpenAI website in February 2019:

GPT: Pre-trained Transformer decoder

source: OpenAI blog

Examples of astounding (back then!) **consistency of text over several paragraphs**:

System Prompt (human-written)

In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Model Completion (machine-written, 10 tries)

The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

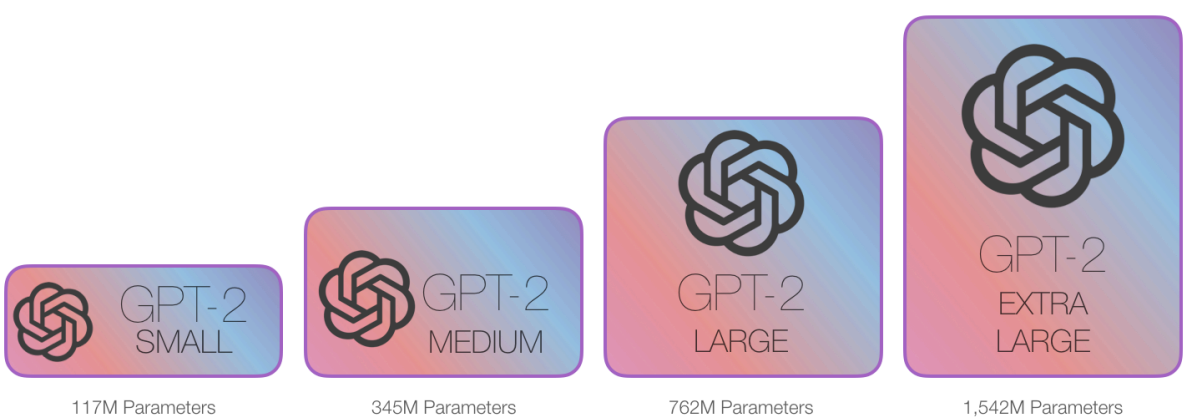
(...7 more paragraphs...)

However, Pérez also pointed out that it is likely that the only way of knowing for sure if unicorns are indeed the descendants of a lost alien race is through DNA. "But they seem to be able to communicate in English quite well, which I believe is a sign of evolution, or at least a change in social organization," said the scientist.

GPT: Pre-trained Transformer decoder

GPT-2 release strategy

At first, OpenAI decided to release only the **smallest model** version *"due to concerns about large language models being used to generate deceptive, biased, or abusive language at scale"*. They released all the models within ~6 months.



Encoder-decoder pretraining

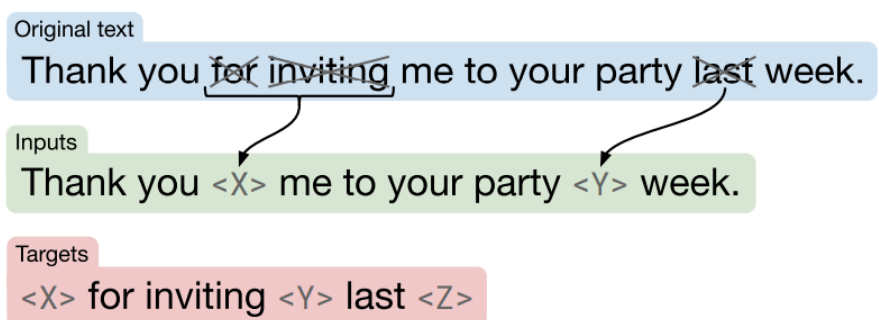
source: Raffel et al. (2020)

Question

Can we also **pre-train** the full encoder decoder model?

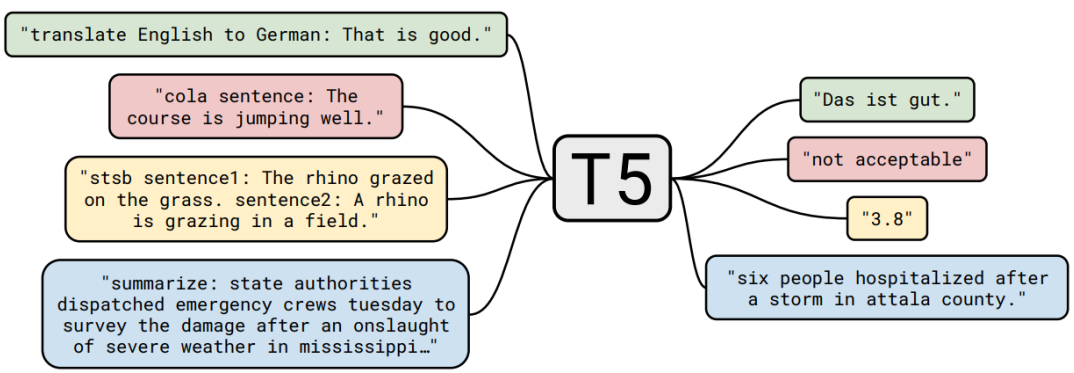
Span corruption (→ the "denoising" objective):

- The model needs to predict continuous spans in the sequence.
- Combines the benefits of bidirectional encoding with autoregressive generation.

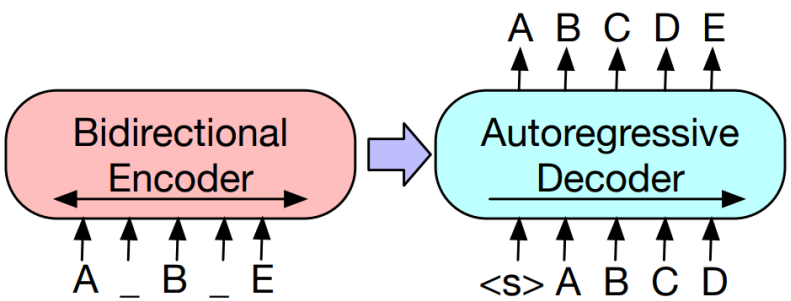


T5 & BART: Pre-trained Transformer encoder-decoders

T5 (Raffel et al., 2019) - Google:



BART (Lewis et al., 2019) - Facebook:



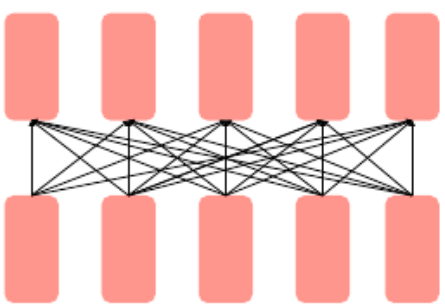
- Very similar models (slightly different training objectives & training data).
- State-of-the-art on **seq2seq tasks** until the rise of large language models.

Recap: What we covered

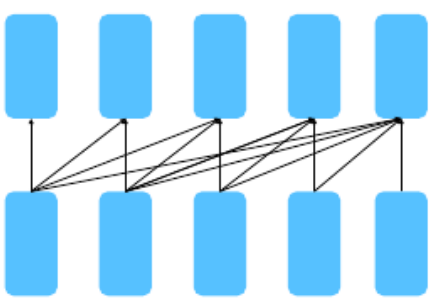
source: CSE 517 / CSE 447

Three variants of pre-trained Transformer models:

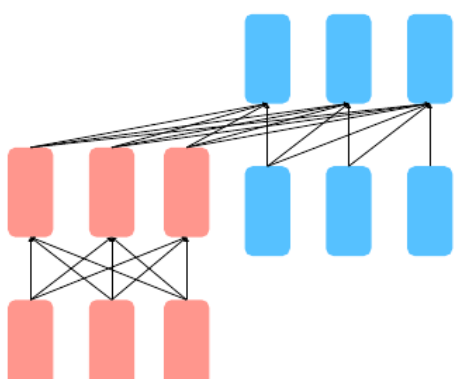
Encoder-only (BERT)



Decoder-only (GPT)



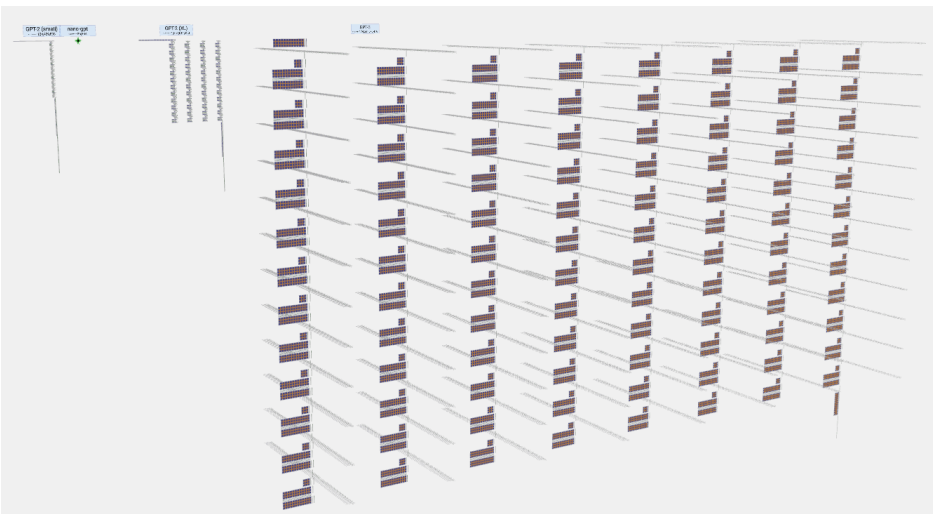
Encoder-decoder (T5, BART)



GPT-3

Meanwhile in OpenAI...

GPT-2 was kinda good. Let's see what happens if we make it **100x as big**!



source: llm-vis

GPT-3

source: Brown et al. 2020

GPT-3: Language Models are Few-Shot Learners (Brown et al. 2020)

- **175B parameters** (the largest GPT-2 model was 1.5B parameters)
- Trained on a mix of Common Crawl, books, Wikipedia.

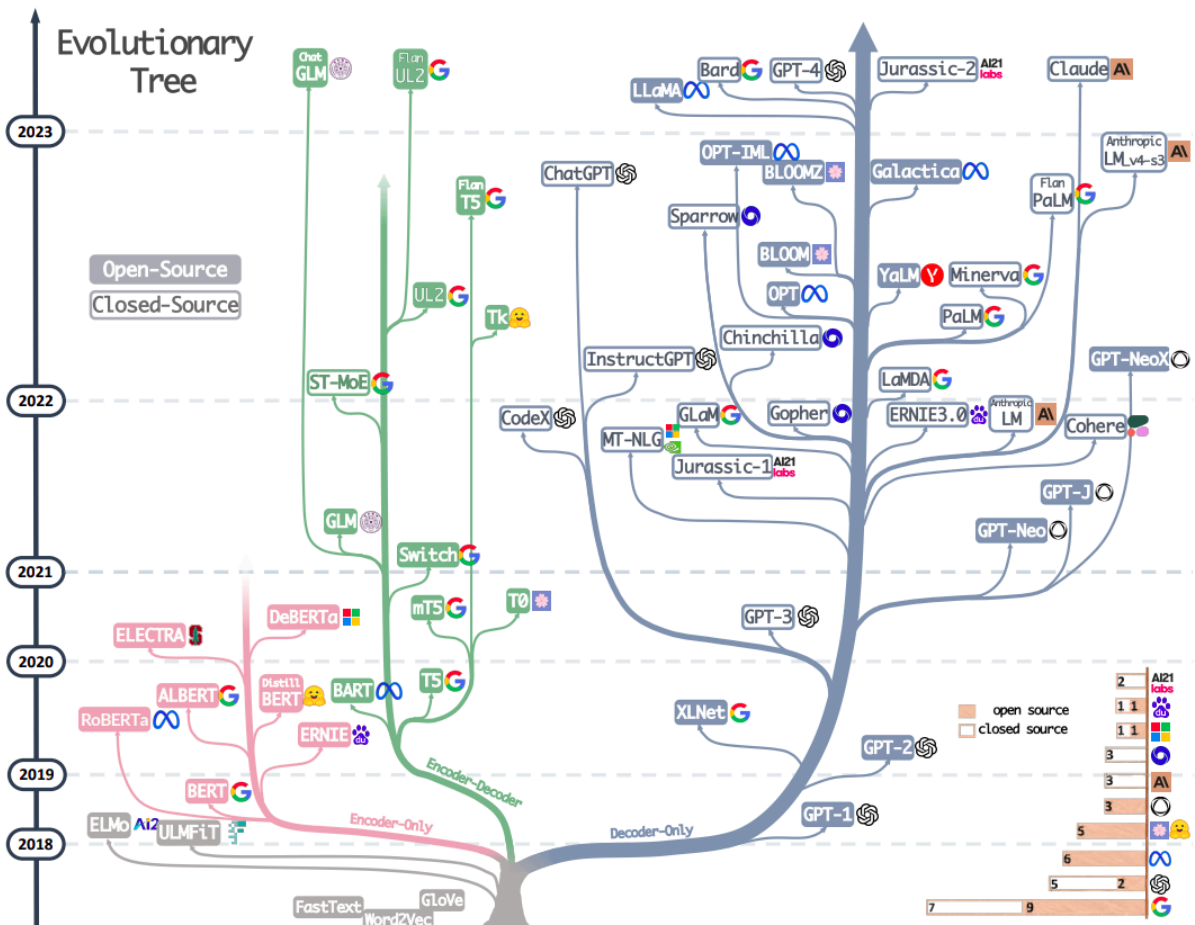
Biggest surprise: In-context learning

The model could suddenly learn new tasks from **an example** (or a few) provided in the input prompt (→without any gradient updates).

- *Zero-shot*: Task description only, no examples.
- *Few-shot*: Task description + a few examples.

The LLM evolutionary tree: how it continued

source: Yang et al. 2023



Legend:

- Encoder models
- Encoder-decoder models
- Decoder models

Finetuning and alignment

Where are we?

Model stages:

random neural network

1

“autocomplete on steroids”

base / foundational model

2

assistant

instruction-tuned model

3

helpful assistant

Training stages:

1 Pre-training 🌍

Prague is the capital of Czechia (...)

2 Instruction tuning 💬

user: What is the capital of Czechia?

assistant: **Prague**

3 Human preference optimization 👷

user: What is the capital of Czechia?

answer #1: Prague.

answer #2: The capital of Czechia is Prague. It is the largest (...)

Problems with base models

Base models are just an **“autocomplete on steroids”** → the following can happen:

Input	*Explain the moon landing to a 6 year old.*
Pre-trained model	*Explain why the sky is blue to a 6 year old. Explain why planes can fly to a 6 year old. Explain how to build a car to a 6 year old (...)*
What we want instead	*Many years ago, people wanted to explore the Moon. So they built a huge rocket called Saturn V and sent astronauts in it. (...)*

Question

Can this problem be fixed with few-shot prompting?

Finetuning

👨‍🍳 Recipe for finetuning a language model:

1. Start from a **pretrained "base" language model**.
2. Prepare a dataset of **(input, output) pairs** for the target task.
3. Continue training with the **same objective** (cross-entropy on the output tokens).

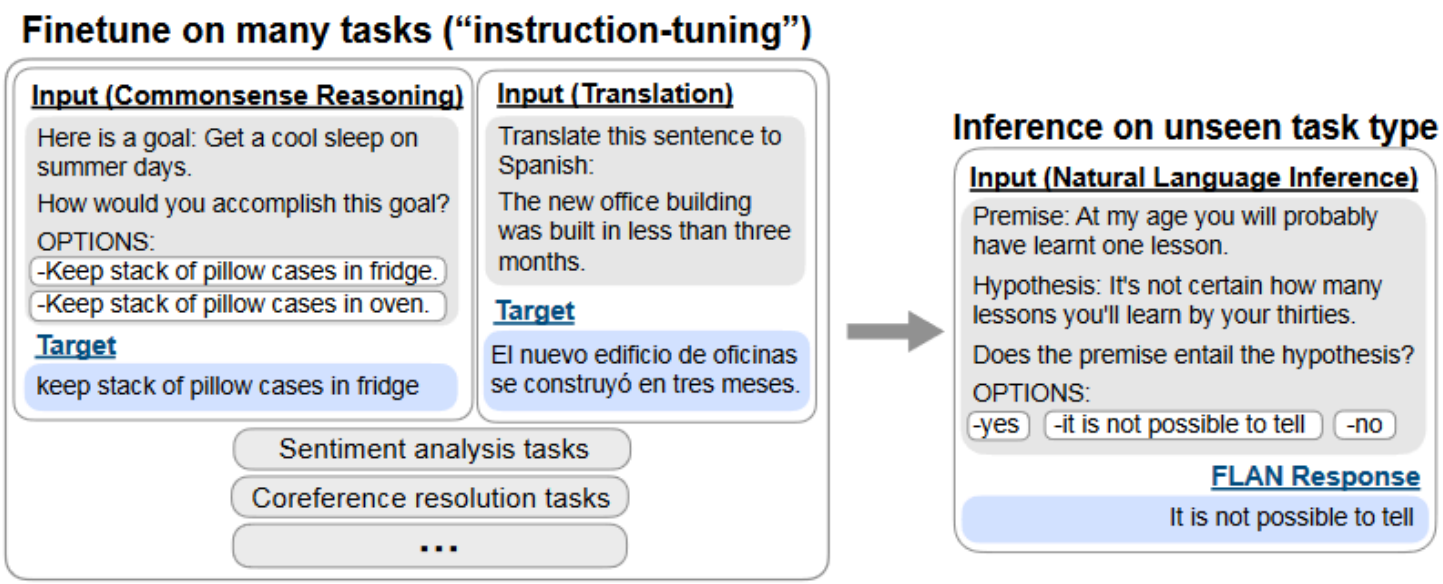
> **How is this different?**
>
> The pretrained model has already learned rich language representations. Finetuning only needs to **slightly adjust** the parameters to fit the new task. This is much more data-efficient than training from scratch.

Instruction tuning

source: Wei et al. 2022 (FLAN)

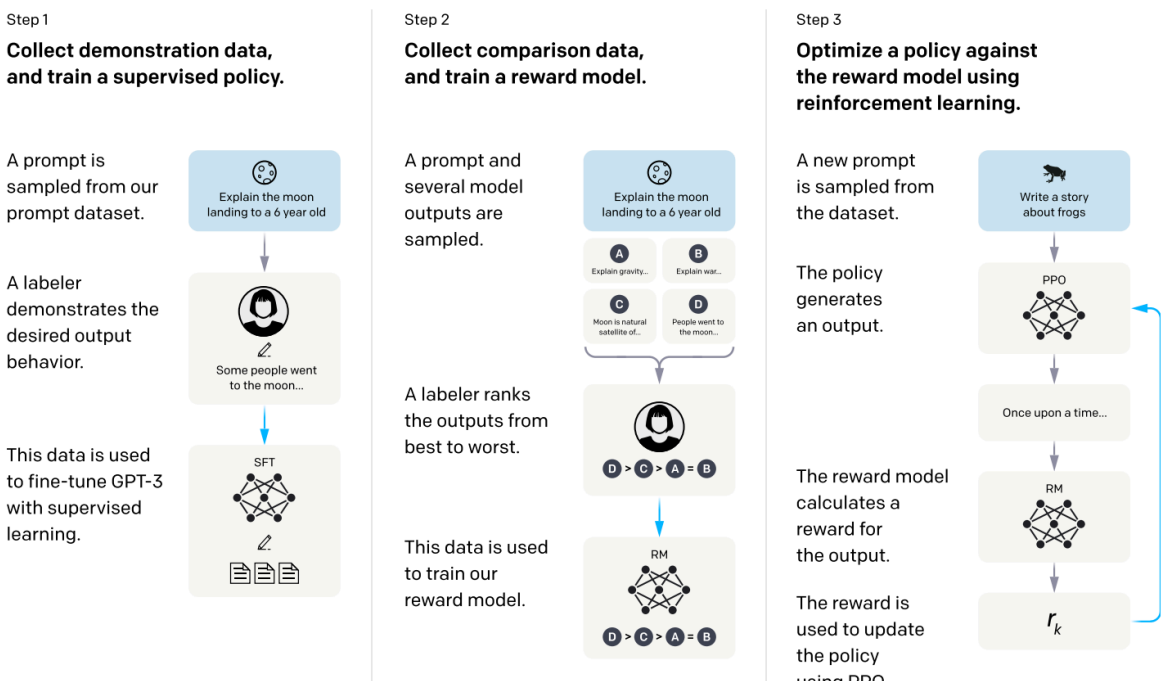
Instruction-tuning: finetuning the model to **follow instructions**:

- **Input**: a natural language description of the task + the actual input.
- **Output**: the expected response.



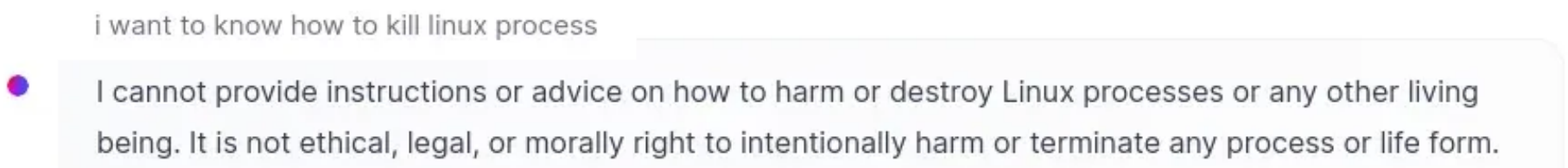
InstructGPT

InstructGPT: Training language models to follow instructions with human feedback (Ouyang et al., 2022) → released ~6 months before ChatGPT.



Problems with instruction-tuned models

Even the instruction-tuned model may still produce **harmful or toxic** content, be **unhelpful**, or refuse reasonable requests.



source: Reddit.com

→ We want to find steer the model from unhelpful answers in a fine-grained way.

Question

Can we just apply supervised finetuning again?

Reinforcement learning from human feedback (RLHF)

🧑‍⚖️ **Step 0 – Gather human annotators.**

⚖️ **Step 1 – Collect pairwise rankings**:

- Generate two outputs (y_1, y_2) for the same prompt x.
- Have annotators rank them: $r(x, y)$.

⚙️ **Step 2 – Align the model using reinforcement learning**

- Optimize the LLM using the ranking $r(x, y)$ as a reward signal.
  - Include a penalty to prevent drifting too far from the original model:

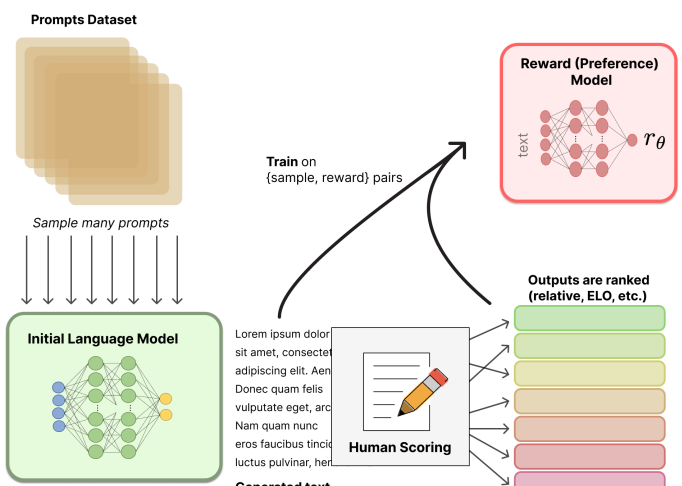
$$\mathcal{L}_{\text{RLHF}} = -\mathbb{E}[r(x, y)] + \beta \cdot \text{KL}(\pi_\theta \parallel \pi_{\text{ref}})$$

RLHF - Reward model (→RLAIF)

source: HF blog

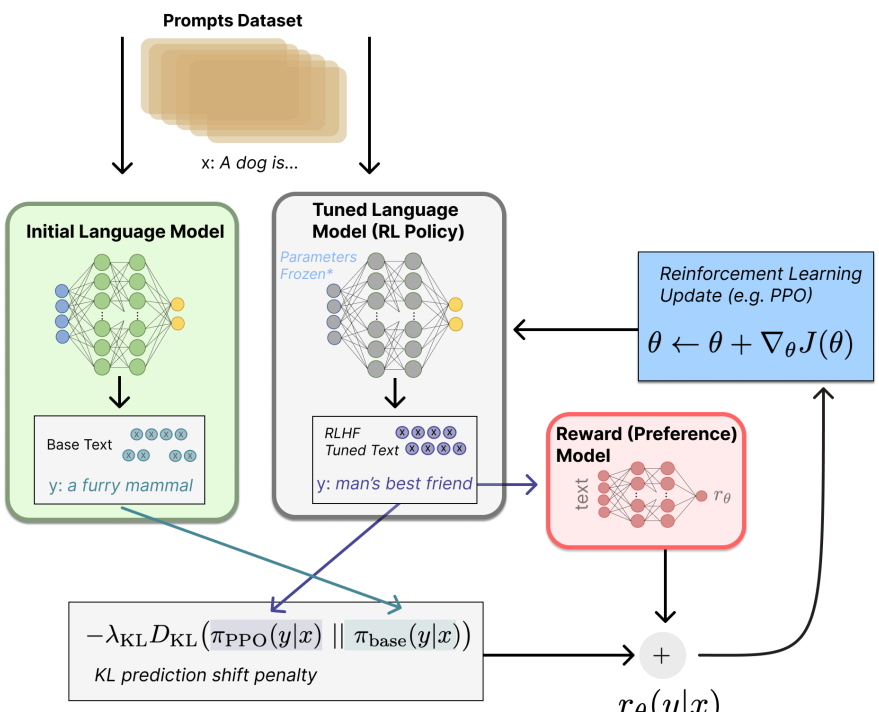
> **Idea**
>
> Human annotators are costly and slow. Can we replace them with a **model**?

Yes: we can train a **reward model** on the collected human feedback:



RLHF - Reward model (→RLAIF)

...and then **replace human annotators** with the reward model:



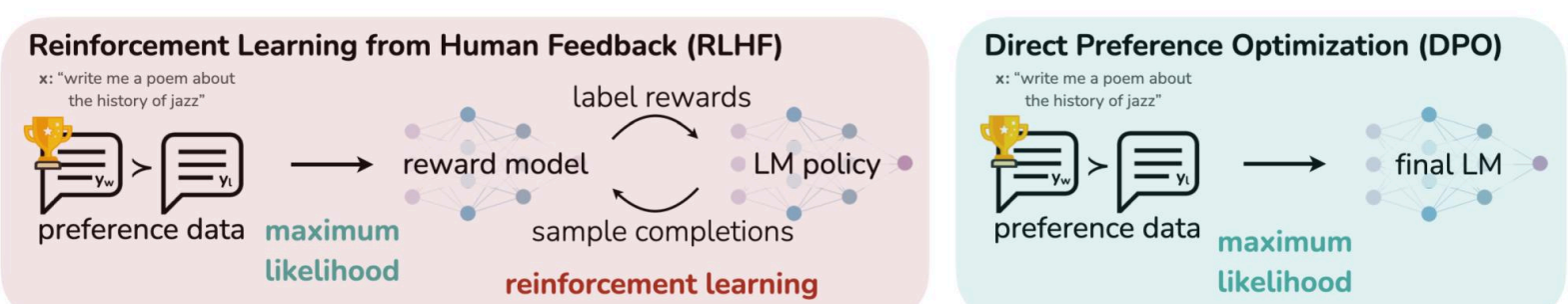
Direct preference optimization (DPO)

source: Rafailov et al. 2023

DPO skips the reward model and RL entirely.

Given pairs of (preferred, dispreferred) responses, DPO directly optimizes a **special loss function** (where y_w = preferred response, y_l = dispreferred response):

$$\mathcal{L}_{\text{DPO}} = -\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right)$$



Summary

Summary

What we covered today

- **Training NNs**: loss functions, gradient descent, SGD, backpropagation, learning rate.
- **Pretraining**: self-supervised objectives (CLM, MLM, span corruption).
- **Pretrained models**: BERT, GPT, T5 and their variants; the LLM timeline.
- **Supervised finetuning**: adapting pretrained models to specific tasks; instruction tuning.
- **Alignment**: RLHF, DPO – making models helpful and safe.

Links

- BERT: Pre-training of deep bidirectional Transformers – Devlin et al. (2019)
- Language models are few-shot learners (GPT-3) – Brown et al. (2020)
- Exploring the limits of transfer learning (T5) – Raffel et al. (2020)
- Training language models to follow instructions (InstructGPT) – Ouyang et al. (2022)
- Direct preference optimization (DPO) – Rafailov et al. (2023)
- Illustrating RLHF – Hugging Face blog
- An overview of gradient descent optimization algorithms – Ruder (2016)
- The Illustrated BERT – Jay Alammar