Fundamentals of Foundation Models - Cheat sheet - Lukas Schüttler - July 22, 2024

## 1 General

**SGD Updates** - $\theta_{k+1} - \theta_k - \sum_{i \in \mathcal{B}} \nabla \operatorname{loss}(f_{\theta_k}, i)$

With $\mathcal{B}$ being a batch os size $B$

**ARM** - Auto regressive model

## 2 Transformers

**Token Embeddings**

$\mathbf{I} \in \mathbb{R}^{\text{voc} \times N}$: Input Sequence - $\mathbf{M}_{\text{emb}} \in \mathbb{R}^{\text{dim\_emb} \times \text{voc}}$: Embedding Matrix - $\mathbf{M}_{\text{pos}} \in \mathbb{R}^{\text{dim\_emb} \times \text{N}}$: Positional Embedding - $\mathbf{E} \in \mathbb{R}^{\text{dim\_emb} \times \text{N}}$: Token Embeddings

$$\mathbf{E} = \mathbf{M}_{\text{emb}} \times \mathbf{I} + \mathbf{M}_{\text{pos}}$$

**Self Attention**

$\mathbf{W} = [\mathbf{W}_q \ \mathbf{W}_k \ \mathbf{W}_v]^\top \in \mathbb{R}^{3\text{dim\_emb} \times \text{dim\_emb}}$ - $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V} \in \mathbb{R}^{\text{dim\_emb} \times N}$: Query, Key, Value - $\mathbf{A} \in \mathbb{R}^{N \times N}$: Attention Matrix - $\mathbf{W}_o \in \mathbb{R}^{\text{dim\_emb} \times \text{dim\_emb}}$: Linear Projection

$$[\mathbf{Q}\ \mathbf{K}\ \mathbf{V}]^\top = \mathbf{W} \times \mathbf{E}$$

$$\mathbf{A} = \operatorname{softmax}\left(\frac{\mathbf{Q}^\top \mathbf{K}}{\sqrt{d_k}}\right) \odot \mathbf{M}_{\text{mask}}$$

$$\mathbf{E}_{\text{att}} = \mathbf{V} \times \mathbf{A} \qquad \mathbf{E}_o = \mathbf{W}_o \times \mathbf{E}_{\text{att}}$$

With $d_k$ being the dimension of the key vectors. E.g. pos_emb for single head attention

**MLP**

$\mathbf{M}_{\text{up}} \in \mathbb{R}^{4\text{dim\_emb} \times \text{dim\_emb}}$: Up projection - $\mathbf{M}_{\text{down}} \in \mathbb{R}^{\text{dim\_emb} \times 4\text{dim\_emb}}$: Down projection

$$\mathbf{E}_{\text{mlp}} = \sigma(\mathbf{M}_{\text{down}} \times \sigma(\mathbf{M}_{\text{up}} \times \mathbf{E}_{\text{att}}))$$

With $\sigma$ being the activation applied elementwise

**Computation:** FLOPS $\approx 6N \cdot D$ - With $D$ being the number of training tokens.

| Operation | Parameters | FLOPs per t. fwd pass |
|---|---|---|
| Input embedding | dim_embd × vocab_size + dim_embd × $N$ | 4 × dim_embd |
| Att: $\mathbf{W}$, $\mathbf{A}$ | $n$_layers × 3 dim_embd$^2$ | 6 $n$_layers × dim_embd$^2$ |
| Att: $\mathbf{E}_{\text{att}}$ | - | 4 $n$_layers × $N$ × dim_embd |
| Att: $\mathbf{W}_o$, $\mathbf{E}_o$ | $n$_layers × dim_embd$^2$ | 2 $n$_layers × dim_embd$^2$ |
| MLP | $n$_layers × 2 × dim_embd × dim_feedforward | 4 $n$_layers × dim_embd × dim_feedforward |
| Embedd | - | 2 dim_embd × vocab_size |
| Total (non-embedding) | $P$ = 2 $n$_layers × dim_embd × (2 dim_embd + dim_feedforward) | 2$P$ + 2 $n$_layers × $N$ × dim_embd |

## 3 Tokenization

**Pre-tokenization**: Split input text via regex to prevent greedy encoding of words in different commen variations: dog? dog!. Additionally tokens lead with a space: ’ *and*’.

### 3.1 Tokenizers

**Byte Pair Encoding (BPE)** - Entropy encoding - Given a base vocabulary size of 256 with UTF-8 BPE encodes additonal pairs of bytes as subsequent numbers to 256 e.g. 257, 258...

Pairs are encoded based on their frequency of occurrence to achieve the maximum compression. The process is iteratively repeated on a vocabulary set until the maximum voc. size is reached (e.g. GPT-4 100k) or the algorithm runs out of pairs.

**WordPiece** - Pointwise mutual information

## 4 Training

### 4.1 Evaluation

**Perplexity**: $\exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(t_i)\right)$

With $t_i$ being the $i$th token in the expected output sequence

**Benchmarks**

**Paloma**: Perplexity over a diverse set to text.

**HellaSwag**: QA Benchmark. Most likely answer by perplexity is chosen.

**MMLU**: QA Benchmark. Answers are part of the Prompt. Model can answer A, B, C or D. Most likely token is chosen.

### 4.2 Datasets

**Common Crawl**: Database of scraped websites. **WebText**: OpenAI internal dataset. Scraped links from Reddit which received at least 3 karma. Page de-duplication and light-cleaning. *8M* Documents, *40GB* of text.

**OpenWebText**: Open replication of **WebText**

**C4**: Colossal Clean Crawled Corpus. Filtered version of **Common Crawl**. Discard pages with fewer than 5 sentences and lines with fewer than 3 words. Filter for unwanted keywords. Remove lines with the word Javascript. Remove pages with "*lorem ipsum*". Remove pages with "*{*". Deduplicate any three-sentence span occurring multiple times. Filter pages that are not in English.

**The Stack**: Coding dataset.

**PeS2o**: STEM papers.

**DOLMA**: Combination of **Common Crawl**, **C4**, **The Stack**, **Reddit**, **PeS2o**, **Project Gutenberg**, **Wikipedia/Wikibooks**

**The Pile**: *800GB* Dataset of Diverse Text. Academic, Internet, Prose, Dialoque and Misc (GitHub, Math ...)

**Dataset Cleaning Pipeline**: Language Filtering → Deduplication (by URL) → Quality Filters → Content Filters → Deduplication (by text overlap)

### 4.3 Fine-Tuning

Possible with around 1000 high quality prompts and responses or more.

**RLHF** - Reinforcement Learning from Human Feedback

$$y_1, y_2 \propto \pi_{\text{SFT}}(y \mid x) \qquad y_w > y_\ell \mid x$$

$$p^*(y_1 > y_2 \mid x) = \frac{1}{1 + \exp(r^*(x \mid y_2) - r^*(x \mid y_1))}$$

$$\mathcal{L}(r) = -\mathbb{E}\left[\log \sigma(r(x \mid y_\ell) - r(x \mid y_w))\right]$$

Where $r$ is the reward model and $\mathcal{L}$ is the loss of the reward model.

$$\max_{\pi} \mathbb{E}\left[r(x, y) - \beta\, \mathrm{D}_{\mathrm{KL}}(\pi(y \mid x) \mid \pi_{\text{ref}}(y \mid x)\right]$$

$\mathrm{D}_{\mathrm{KL}}$ to reduce the deviation from the base model (SFT).

**DPO** - Direct Preference Optimization

$$\mathcal{L}_{\text{DPO}} = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_\ell \mid x)}{\pi_{\text{ref}}(y_\ell \mid x)}\right)\right]$$

## 5 Ensembles and MoE

### 5.1 Ensembles

**Linear interpolation** - $P(y \mid x) = \sum_m P_m(y \mid x) P(m \mid x)$

**Log-linear interpolation** - $\operatorname{softmax}\left(\sum_m \log P(y \mid x) \lambda_m(x)\right)$

With $P_m(y \mid x)$ being the output of the model $m$ and $P(m \mid x)$ the "reliability" of the model given the Input. $\lambda_m$ is an interpolation coefficients for the model $m$.

**Parameter Averaging** - Calculate model weights by accumulating (average, weighted average ...) weights from multiple models

### 5.2 MoE - Mixture of Experts

**Gaussian mixture model** - $p(y \mid x) = \sum_k p_{\theta_k}(y) p_k(x)$
Where $p_{\theta_k}$ is a gaussian distribution parametrized by $\theta_k$ giving the propabillity of the output $y$. $p_k$ is the probability of that distribution given the input $x$.
Allows the approximation of more complex distributions using only simple distributions (gaussian and logistic)

**Routing** - $\mathbf{W}_r$: Router weights
**Shazeer** - $\mathbf{G}(x) = \text{softmax}(\mathbf{W}_r \mathbf{x})$, $\mathbf{y} = \sum_{k \in \text{top}-\text{k}} \mathbf{G}_k(x) \mathbf{E}_{\text{mlp}}(\mathbf{x})$
**Mixtral** - $\mathbf{G}(x) = \text{softmax}(\text{topK}(\mathbf{W}_r \mathbf{x}))$, $\mathbf{y} = \sum_k \mathbf{G}_k(x) \mathbf{E}_{\text{mlp}}(\mathbf{x})$
**Switch routing** - Improves distributed training, Batch is split between experts, If expert reaches maximum tokens remaining tokens are passed through without calculation

## 6 Scaling Laws

$$L(D) \approx \frac{A}{D^{\alpha_D}} \qquad L(N) \approx \frac{B}{N^{\alpha_N}}$$

$$R(f_{N,D}) = R(f^*) + (R(f_N) - R(f^*)) + (R(f_{N,D}) - R(f_N))$$

$$R(f_{N,D}) = E + L(N) + L(D)$$

Let $R(f^*)$ be the irreducible error, $R(f_N) - R(f^*)$ the approximation error of a N-parameter model and $R(f_{N,D}) - R(f_N)$ the statistical error
Parameter Values depend on data and precise model architecture.

### 6.1 Compute Optimality

$\arg\min L(N, D)$ for a constant compute budget $C(N, D) = H$ by choosing the best $N$ and $D$.
**Training curve envelope** - Plot training curves ($x$: log(FLOPS), $y$: $L(N, D)$), Find the envelope (Minimal loss per FLOP), Plot envelope points twice ($x$: log(FLOPS), $y$: $\log(D)$, $\log(N)$), Fit a line to the points and extrapolate to the desired FLOPS to find the optimal $N$ and $D$.
**IsoFLOP Curves** - Train various model sizes with $D$ such that the final FLOPs are constant, Repeat for different final FLOPs, Plot final loss (x: $\log(N)$, y: $L(N, D)$), Locate optimal model size for a given compute budget (loss valley), Plot optimal models ($x$: log(FLOPS), $y$: $\log(D)$, $\log(N)$) and extrapolate.
**Parametric fit** - Fit parametric Risk function $R(f_{N,D})$ to the training results (Training like IsoFLOP), Plot contours ($x$: log(FLOPS), $y$: $\log(N)$), Fit line such that it goes through each iso-loss contour at the point with the fewest FLOPs. Extrapolate to desired FLOPs. Alternative - Plot isoFLOP slice (x: $\log(N)$, y: $L(N, D)$), plot parametric risk for desired compute budget, Locate the minimum.

**Key finding:** For compute optimal training $D$ should scale proportionally with $N$ - Compute Optimality doesn't consider inference cost

## 7 Scalable Computing

### 7.1 Multicore Processing

GPUs are optimized for performing the same operation on different data simultaneously (**SIMD** - single-instruction multiple-data)
**Amdahl's law** - Let the non-parallizable part of a program take a fraction $s$ of the time, then $m$ workers can result in a speedup of:

$$\frac{1}{s + \frac{1-s}{m}}$$

**Floating Point Numbers**

| | Sign | Exponential | Mantissa |
|---|---|---|---|
| **FP32** | 1 Bit | 8 Bits | 23 Bits |
| **FP16** | 1 Bit | 5 Bits | 10 Bits |
| **BF16** | 1 Bit | 8 Bits | 7 Bits |

**BF16** - **FP32** range with **FP16** precision
**Error:** $\mid x - \widetilde{x} \mid \leq \epsilon/2 \mid x \mid$ with $\epsilon \neq 0$

### 7.2 CUDA

**CUDA Thread** - Smallest compute entity
**CUDA Block** - Group of threads (up to 1024)
**Streaming Multiprocessor** - Executes one **CUDA Block**

### 7.3 Distributed Computing

**Communication Patterns**
**Push** - $A$ sends data to $B$
**Pull** - $B$ requests data from $A$
**Reduce** - Data from $C_1, \dots C_n$ is aggregated on $A$ (sum, average, min, max ...)
**All-Reduce** - Reduce but result is distributed back to $C_1, \dots C_n$
**Scatter-Reduce** - ...
**Wait** - $A$ waits fo a signal from $B$
**Barrier** - $C_1, \dots C_n$ wait until all $C_i$ reach a point of execution and then continue

**Training Methods**
**Synchronous** - Each machine computes a minibatch. Gradients are aggregated and redistributed. Machines are stalled until redistribution
**Asynchronous** - Each machine computes minibatch. Gradients are aggregated and redistributed when possible. Machines continue with next minibatch before redistribution
**Model parallel** - Model layers are distributed between machines (*slower, than running on a single GPU*)
**Model parallel Pipeline** - **Model parallel** but when 2nd GPU processes 1st micro-batch 1st GPU starts with the 2nd one ...Backwards pass starts from last GPU when all micro-batches are done.

## 8 Vision Models

**Segmentation** - Extract patch, classify center pixel - Fully convolutional network can classify all pixels at once, up-sampling
**Upconvolution** - Fractional Strided Convolution
**U-Net** - Down-sampling + Max pooling for classification, Up-sampling + Skip connection to regain detail

### 8.1 Autoencoders

Project input into low dimensional (latent) space $\mathcal{Z}$ so that original data can be recovered.
**VAE** - Variational Autoencoder
$Z$ has a known distribution $p_z(z \mid x)$ (normal distribution), $p_z$ is forced to be close to $\mathcal{N}(0, \mathbf{I})$ so that distributions of different $x$ overlap.
**Denoising Autoencoder** - High dimensional latent space would usually lead to identity $\implies$ parts of the input or latent representation are removed or noise is added to force robust representation.

### 8.2 Generative Models

**GAN** - Generative Adversarial Network - *PRO:* high-quality, fast - *CON:* Implicit density, training unstable, only external measures for comparing
**VAE** - *PRO:* Explicit density, fast - *CON:* lower bound on density, coverage / quality trade-off
**Diffusion Models** - *PRO:* flexible architecture, stable training, latent variable, high reselution - *CON:* Expensive due to long markov chain

$$q(x_t \mid x_{t-1}) = \mathcal{N}(\sqrt{1 - \beta_t} x_{t-1}, \beta_t \mathbf{I})$$