References

- Switch Transformer (https://arxiv.org/pdf/2101.03961.pdf)
- Towards Understanding Mixture of Experts in Deep Learning (https://arxiv.org/pdf/2208.02813.pdf)

Scaling Laws

In our module on [Transformer Scaling (Transformers_Scaling.ipynb)] we learned

- that a model with more parameters (denoted N)
- is more example-efficient
    - for a given loss level L: a large model achieves L with fewer examples than a smaller model
- but at the cost of greater computation C
    - each parameters is consumed in some computation

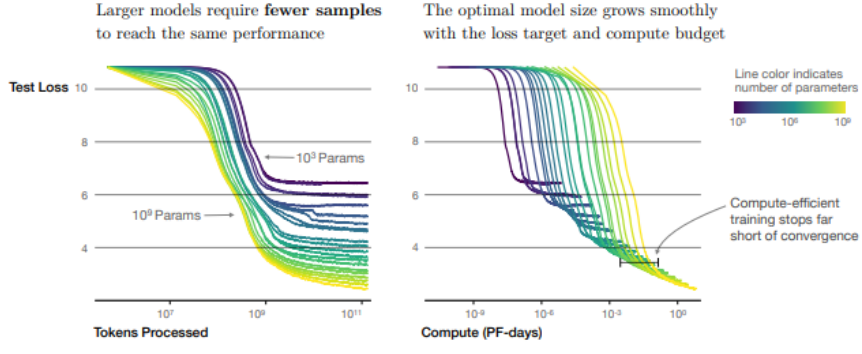

Figure 2 We show a series of language model training runs, with models ranging in size from 10^3 to 10^9 parameters (excluding embeddings).

Attribution: https://arxiv.org/pdf/2001.08361.pdf#page=4

This module describes a technique to facilitate

- large models (greater N)
- without increasing computation C

Mixture of Experts (MoE)

Let us consider a binary classification task

- given an example $(\mathbf{x}, \mathbf{y})$ where
  - $\mathbf{x}$ is from distribution $\mathcal{D}$
  - $\mathbf{y} \in \{0, 1\}$
- compute $p(\mathbf{y} = 1 | \mathbf{x} \in \mathcal{D})$

We would typically solve this task by constructing a Neural Network with N parameters Θ

- computing $p_\Theta(\mathbf{y} = 1 | \mathbf{x} \in \mathcal{D}) \approx p(\mathbf{y} = 1 | \mathbf{x} \in \mathcal{D})$

The idea of a *Mixture of Experts (MoE)*

- is to use a union of M distinct neural networks
- where Neural Network with index m is an *expert* in solving the tasks for examples in a subset of $\mathcal{D}$

Neural Network with index m

- has its own set of parameters Θ_m, ($0 \leq m \leq M-1$)
  - each of size N
- specializing in classifying a subset $\mathcal{E}_m$ of $\mathcal{D}$

$$p_{\Theta_m}(\mathbf{y}=1|\mathbf{x} \in \mathcal{E}_m)$$
$$\cup_{m=1}^{M} \mathcal{E}_m = \mathcal{D}$$

It would seem plausible that the Mixture of Experts

- would have lower loss on approximating $p(\mathbf{y} = 1 | \mathbf{x} \in \mathcal{D})$
- than the single model p_Θ

Unfortunately: this comes at the cost of a factor of M more parameters.

For example, Suppose that the $\mathbf{x} \in \mathcal{D}$ naturally form C clusters $\{\mathcal{D}_1, \ldots \mathcal{D}_C\}$

$$\mathcal{D} = \cup_{c=1}^{C} \mathcal{D}_k$$

- e.g., each cluster is a "topic" (Business, Arts, Science) or language (English, French, Mandarin)

Lettng

- $M = C$
- $\mathcal{E}_m = \mathcal{D}_m$
- we devote one expert per cluster

In order to make this idea complete, we hypothesize a *router*

- that computes the probability that example $\mathbf{x}$ is best suited to Expert m
- computes a vector of length M
$$p_0(\mathbf{x}), \ldots, p_{M-1}(\mathbf{x})$$
- such that
$$p_m(\mathbf{x}) = p(\mathbf{x} \in \mathcal{E}_m)$$

The idea would be to "route" example $\mathbf{x}$ to the expert(s) best suited for its classification.

The Mixture of Experts network could compute the probability weighted sum of the experts

$$\sum_{m=0}^{M-1} p_m(\mathbf{x}) * p_{\Theta_m}(\mathbf{y}=1|\mathbf{x})$$

Mixture of Experts Layer

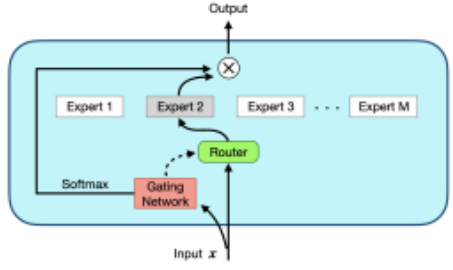

Figure 2: **Illustration of an MoE layer.** For each input **x**, the router will only select one expert to perform computations. The choice is based on the output of the gating network (dotted line). The expert layer returns the output of the selected expert (gray box) multiplied by the route gate value (softmax of the gating function output).

Attribution: https://arxiv.org/pdf/2208.02813.pdf#page=6

In an idealized world

- the vector would have a single element m' with
$$p_i(\mathbf{x}) = \begin{cases} 1 & i = m' \\ 0 & i \neq m' \end{cases} \text{ where } \mathbf{x} \in \mathcal{E}_{m'}$$
- we could somehow avoid computing $p_{\Theta_m}(\mathbf{y} = 1|\mathbf{x})$ for experts with 0 probability $p_m(\mathbf{x})$

But, in a non-idealized world where each expert has a non-zero $p_i(\mathbf{x})$

- the amount of computation is M times that of the single Neural Network
- since each expert m must compute $p_{\Theta_m}(\mathbf{y}=1|\mathbf{x})$

In summary: the naive MoE model increases (by a factor of M) both

- parameters
- computation

A way to increase

- number of parameters by factor of M
- but increase computation by a *constant* factor $k \geq 1$
- is to sort the probabilities

$$p_{m_0}(\mathbf{x}) \geq p_{m_1}(\mathbf{x}) \geq \ldots \geq p_{m_{M-1}}(\mathbf{x})$$

- and route $\mathbf{x}$ to only the *top-k* experts
  - $T_x = \{m_0, \ldots, m_{k-1}\}$
- normalizing the k probabilities to sum to 1

Thus we

- have a bigger model (factor of M)
- with constant multiplicative increase in computation

When $k < M$, some of the experts are not activated for each example.

We call such a model *sparsely activated* (as opposed to *densely activated*).

Training an MoE model

The idea sounds simple enough

- construct a sub-network for the router, computing
$$p_0(\mathbf{x}), \ldots, p_{M-1}(\mathbf{x})$$
- construct M sub-networks of identical experts (with *separate* weights)
- jointly train the model on a standard training set of examples $\langle \mathbf{x}, \mathbf{y} \rangle$

We just jointly train the sub-networks to optimize the Loss function.

- See (https://arxiv.org/pdf/2208.02813.pdf#page=6)

As is the usual practice with Neural Networks

- we are not pre-determining
    - how the router decides which expert is best for a given $\mathbf{x}$
    - which subset $\mathcal{E}_m$ the expert with index m decides to specialize on

It is somewhat of a miracle that joint training works.

For instance, the MoE model could collapse to using a single expert m' for all examples

- router ignores $\mathbf{x}$ and computes
  $p_i(\mathbf{x}) = \begin{cases} 1 & i = m' \\ 0 & i \neq m' \end{cases}$ for all $\mathbf{x}$

This would seem to be an issue in the early epochs

- experts have initialization from same distribution
    - each has expected classification probability of $p_{\Theta_m}(\mathbf{y} = 1) = 0.5$
    - minor differences not significant
    - but may influence the router

A related issue should be familiar to those familiar with Reinforcement Learning

- trade-off between exploitation and exploration
    - router would favor highest probability expert, even when probability differences are insignificant
        - *exploitation*
    - router might learn of a better routing by only choosing *non-highest-rated expert* with some probability
        - *exploration*

The [hard switching (https://arxiv.org/pdf/2101.03961.pdf#page=8)](https://arxiv.org/pdf/2101.03961.pdf#page=8) of the router can also cause instability

- an insignificant difference in routing probabilities $p_i(\mathbf{x})$ and $p_{i'}(\mathbf{x})$ causes $\mathbf{x}$ to *always* be routed to expert i
- soft choice distributes the probability almost evenly between i and i'

In practice, methods to avoid this includes

- sample from the argmax (https://arxiv.org/pdf/2101.03961.pdf#page=29) (that determines the top-k experts) rather than using deterministic choice
- add noise (https://arxiv.org/pdf/2208.02813.pdf#page=6) to $p_i\left(\mathbf{x}\right)$

Another issue

- the `argmax` (used in choosing top-k)
    - is a hard choice
    - not a soft choice (as in using a sigmoid to implement a soft (http://localhost:8888/notebooks/NYU/Neural_Programming.ipynb#%22I statements---Gates) if-then-else
    - so is not differentiable

I believe this is not an issue

- since the gradient that would be routed to an inactive expert would be 0 in any event

---

Switch Transformer

The Switch Transformer uses a Mixture of Experts

- to replace the Feed Forward Network (FFN) component of a Transformer
- every few Transformer blocks

It chooses $k = 1$

- routing to a single expert

Switch Transformer Encoder

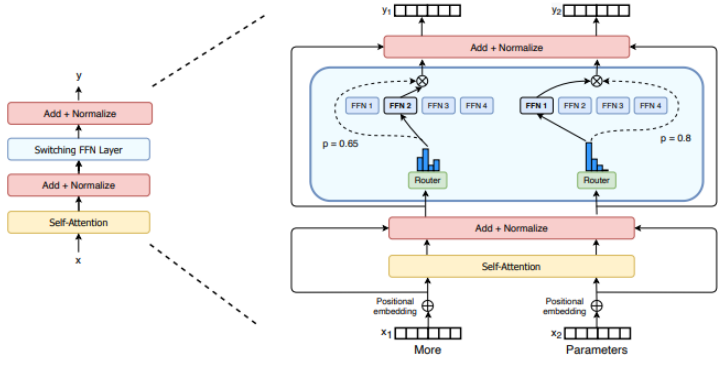

Figure 2: Illustration of a Switch Transformer encoder block. We replace the dense feed forward network (FFN) layer present in the Transformer with a sparse Switch FFN layer (light blue). The layer operates independently on the tokens in the sequence. We diagram two tokens (x_1 = "More" and x_2 = "Parameters" below) being routed (solid lines) across four FFN experts, where the router independently routes each token. The switch FFN layer returns the output of the selected FFN multiplied by the router gate value (dotted-line).

Attribution: https://arxiv.org/pdf/2101.03961.pdf#page=5

Optimization

The [Switch Transformer (https://arxiv.org/pdf/2101.03961.pdf)](https://arxiv.org/pdf/2101.03961.pdf) paper is very concerned with maximizing computational potential.

Suppose the M experts are distributed over several computational units (cores or processors)

- if too many tokens in a mini-batch are routed to the same expert m
- expert m becomes a bottle-neck
- while all the computational units of the other experts $m' \neq m$ are idle

Their solution

- define a *capacity* for each expert
  - number of buffers allocated to each expert: maximum number of tokens it c
- implement differentiable load-balancing (https://arxiv.org/pdf/2101.03961.pdf#pag
  - add a term to the Loss that penalizes for a non-uniform distribution of toke

$$
\begin{aligned}
f_m &= \frac{1}{T} \sum_{\mathbf{x} \in \text{batch}} 1 * (\text{argmax}(p(\mathbf{x})) = m) && \text{fraction} \\
P_m &= \frac{1}{T} \sum_{\mathbf{x} \in \text{batch}} p_m(\mathbf{x}) && \text{fraction} \\
\mathcal{L}_{\text{load balance}} &= \alpha * M * \sum_{m=0}^{M-1} f_m * P_m && \text{minimize} \\
& && \alpha \text{ is relat} \\
& && \text{multiply}
\end{aligned}
$$

- Note (https://arxiv.org/pdf/2101.03961.pdf#page=8)
  - P-vector is differentiable, f vector is not

This is an example of

- practical engineering, supported by math
- rather than theoretical perfection

In [2]:
```
print("Done")
```

```
Done
```