The reasoning of LLM's is currently limited in a fundamental way: the easiest token to predict takes as much compute as the hardest token to product. Another way to think about it is that the flow of information changes when conditioned on different inputs, but not the size of the flow. It is very easy to design a network that can change in size with respect to the input. But no one has found one that can learn effectively from gradient descent, due to the shape rigidity imposed by the need for differentiability of the network.

Here, I propose a rather primitive attempt at doing so. This prototype assumes familiarity with current autoregressive and token-based LLM's. These LLM's have only tokens that represent bits of natural langage. The proposed prototype has an additional set of hidden tokens that are meant to represent bits of thoughts. Current LLM's are used autoregressively to generate sentences. My prototype is also used autoregressively to produce a single word. At each time step, it always produces a guess of the next word token, but it also produces a new bit of thought that could help it produce an even better guess of the next word token. Its input context is also split into two parts: the question context (X) and the reasoning context (H). It achieves reasoning by modifying H with each forward pass.

Here what 3 reasoning steps would look like:

$$\begin{tikzcd}
 & \hat{Y}_1 & & \hat{Y}_2 & & \hat{Y}_3 \\
(X, H_1) \arrow[r] & N \arrow[u] \arrow[r] & (X, H_2) \arrow[r] & N \arrow[u] \arrow[r] & (X, H_3) \arrow[r] & N \arrow[u]
\end{tikzcd}$$

It is very important to note that all the $\hat{Y}'s$ are guesses (vectors of probabilities) for the **same** Y.

The autoregressiely added hidden token added at each time step is $h_{t+1} = \sum_i p_i^t h_i$ where p^t is the softmax of the output neurons associated to the hidden tokens at time t. Thus the network outputs a probability over the set of output tokens **and** hidden tokens. Let S_h be the total amount of probability that the model attributes to the hidden tokens at time step t. Let $S_{\hat{y}} := 1 - S_h$ be the total amount of probability that the model attributes to word tokens. Assuming we have a loss function L for the word tokens probabilities $\hat{Y}$ and the ground truth Y, (say, the one used for GPT3), then we define the loss for the output neurons corresponding to the $\hat{Y}_t$ tokens to be $L_{\hat{y}}^t := S_{\hat{y}}^t \cdot L(\hat{Y}_t, Y)$ and the loss for the output neurons corresponding to the hidden tokens to be $L_h^t := S_h^t \cdot [L_{\hat{y}}^{t+1} + L_h^{t+1}]$. One must decide at which t to set $L_h^t = 0$.
Afterthought: perhaps the size of the gradient step for the loss of a neuron should be proportional to the probability of reaching it.

2024-03-14