Faster Convergence for Transformer Fine-tuning with Line Search Methods

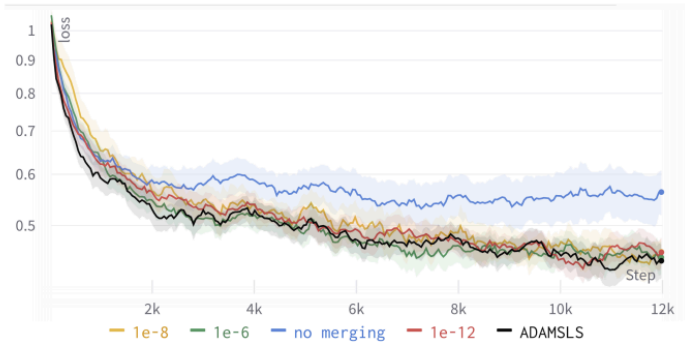

Fig. 2. Different merging thresholds on one epoch of the MNLI dataset. Standard error is indicated around each line.

It's been a while since I wrote a review, and I came across an awesome paper discussing a family of approaches for training neural networks that I was not aware of. You probably know how today deep neural networks and other ML models are trained (i.e., their loss function is minimized). This is, of course, done with various refinements of the stochastic gradient descent (SGD) approach.

In general, SGD is an iterative method where, in each iteration, the model's weights are shifted in the direction of the negative gradient, which is the (linear) direction in which the loss function decreases "the most." As mentioned, there are quite a few refinements of SGD, such as ADAM, RMSProp, and many other methods that involve gradient accumulation (momentum), aimed at accelerating the convergence rate of SGD and making it more stable. It's worth noting that in all these methods, the weights are updated in each iteration based on a mini-batch rather than a single example, as in classical SGD.

The paper we will review today proposes a different (faster, according to the authors) approach to minimizing the loss function for transformer-based models. First, note that the value of the loss function does not necessarily decrease (on the mini-batch of the current iteration) after updating the model parameters in each iteration of any gradient descent-based method.

Sometimes the loss on the mini-batch may increase after the update, even if you are using advanced methods like ADAM or RMSProp. For SGD (i.e., MiniBatch GD), this generally happens because the learning rate is too high. In momentum-based methods like ADAM, the situation is more complex (since the direction in which the weights are moved is not the average gradient of the mini-batch), but the problem still exists.

It's important to understand that a temporary increase in the loss function for mini-batches here and there is "not a disaster" if the overall trend of loss reduction is maintained during training. But the question arises: would a training method that ensures the loss function never increases in any iteration lead to more efficient and faster training without compromising the final model quality? This is the question the authors of the paper attempt to answer.

The method discussed in the paper offers a very intuitive way to circumvent this issue. As mentioned, the non-decrease of the loss function in SGD is due to a too-high learning rate. The paper suggests choosing the learning rate so that it ensures a decrease in the loss function for every mini-batch in every training iteration.

In general, in each training iteration, you start with a random learning rate and reduce it (say, by dividing by 2) until the loss after the update decreases (on the mini-batch). This can also be done with the advanced methods mentioned above, where the final step is moving the weights in a certain direction with a certain coefficient (learning rate). This method is called Armijo Line Search or ALS.

The paper suggests applying ALS to each layer (transformer block) separately. In other words, the model's weights are divided into L groups, where L is the number of layers in the network. Then, the weights for each layer are updated separately using the optimization method of your choice (SGD, ADAM, etc.) combined with ALS. That is, the learning rate for each layer is reduced separately until the value of the loss function after the update decreases, while the other weights are frozen.

In my opinion, with the proposed method, the training will take longer (since each layer is updated separately), but it allows for working with larger batches, which contributes to the stability of the training process. The method shows decent results in fine-tuning transformers.