Expert Gate: Lifelong Learning with a Network of Experts

Main idea: this paper experiments with a novel approach for using MoE in a multi-task setting. More specifically, it focuses on the gating mechanism used. Expert Gate also focuses on scalability, as it is a lifelong learning approach (can be scaled with time). A lifelong learning approach means that:

- Models are trained sequentially.
- No need to store the data used for training, only the models.

Expert Gate is trained on image classification and video prediction problems, but could technically also be used in an NLP/LLM setting (but was not experimented with)

Advantages of Expert Gate

The meat of this method is in the autoencoder gating mechanism used. This mechanism solves problems as:

- Data storage, since the models can be trained sequentially, so keeping all training data is not necessary.
    - Later the paper will show that storing training data used previously is not necessary.
- Catastrophic forgetting, which is an issue other models suffer with. For example, continuously training and fine-tuning the same model on new tasks will lead to this issue.

- Task biases when fine-tuning which can lead to suboptimal local minima.
    - If a model is trained on a task and fine-tuned on a widely different task, it can lead to suboptimal results due to the biases inferred in the initial task being different.
- Memory efficiency, as only one expert needs to be loaded into memory at a time.
- Task relatedness, which can be measured by the autoencoder's results and then be used to figure out how to initialize the expert's parameters for a new class and either to use fine-tuning or learning-without-forgetting (LwF) for training the new expert.

LwF vs Fine-tuning

When two tasks are sufficiently related (above a certain task relatedness in threshold), it is beneficial to train a new expert with LwF based on an old task, otherwise the best approach is to fine-tune the expert for the similar (existing) task on the training data from the new task.

- Fine-tuning
    - Based on an existing model, simply continue training using a new dataset
    - The result of this fine-tuning on an existing expert will be a brand-new expert, while the existing expert that it was based on will remain unchanged.
        - So, this process starts with 1 expert and ends up with 2 experts.
- LwF
    - Technique used to prevent catastrophic forgetting when training an existing model on new data. LwF uses soft targets (outputs of the old model) to help retain knowledge from old tasks.
    - As with fine-tuning, this results in 2 experts.

Autoencoder Mechanism – Expert Gate Inner Working

Goals:

- To select an expert based on input data.
- To measure task relatedness to figure out optimal parameters to initialize an expert (based on most related task) and training strategy (fine-tuning or LwF – LwF in the case of task relatedness being above a certain threshold).

The Inner Workings of the Autoencoder

- It follows a regular encoder-decoder architecture.
- Encoder $f(h(x))$, maps the input x to a code h(x).
- Decoder $r = g\big(h(x)\big)$, maps the encoder's code (h(x)) to a reconstruction of the input.
- The autoencoder simply uses an encoder-decoder architecture to deconstruct the input (done by the encoder) and reconstruct it (done by the decoder).
- The loss function $L\left(x, g\big(h(x)\big)\right)$ is simply the reconstruction error.
- The encoder learns, through a hidden layer, a lower dimensional representation (undercomplete autoencoder) or a higher dimensional representation (overcomplete autoencoder) of the input data.
- The lower dimensional subspace learned by one of the undercomplete autoencoders will be maximally sensitive to variations observed in the task data but insensitive to changes

orthogonal to the manifold (it represents only the variations that are needed to reconstruct relevant samples)

  - The autoencoder of a domain/task should be better at reconstructing the data of the task it was trained on better than the other autoencoders.
    - The reconstruction error for each autoencoder then allows the input to be routed to the expert of the task of the autoencoder with the lowest reconstruction error for that input (or multiple, in the case of multiple very good autoencoders for that input).
    - The reconstruction error then acts like a score (all reconstruction errors are passed through a SoftMax to determine a normalized score).

- The task relatedness between two tasks is also measured through the autoencoder's reconstruction error through the following formula:
  - $Rel(T_k, T_a) = 1 - (\frac{(ERR_a - ERR_k)}{ERR_k})$
    - T_k = new task. T_a = old task.
    - $Rel(T_k, T_a)$ is the relatedness between task k and task a.
    - ERR_a is the reconstruction error of the autoencoder for task a in the data for task k.
    - ERR_k is the reconstruction error of the autoencoder for task k on its own data.
    - How can the reconstruction error of the autoencoder for task k on its own data be computed before the expert (and thus its autoencoder) is trained,

since its initialization method relies on this task relatedness computation? This seems redundant.

Experiments Results

- Expert Gate was compared with and outperformed (on image classification):
    - Single fine-tuned model (sequentially fine-tuned on each task).
        - One would think that this would result in severe catastrophic forgetting.
    - Single LwF model (sequentially trained on each task).
        - One would think that you can't train the same model with LwF forever on many different tasks without running into catastrophic forgetting issues.
- Expert Gate performed on-par with:
    - Joint training (assumes all is always available for re-training).
    - Multiple fine-tuned models (fine-tuned on each task separately)
        - This assumes an oracle gate, that is, a gate that knows perfectly how to route each input to the corresponding expert.
    - Multiple LwF models (trained on each task separately).
        - Also assumes an oracle gate.
- Expert Gate vs Discriminative Classifier (neural net trained on all the data available for gating decisions – a routing mechanism).
    - Without ever having simultaneous access to the data of different tasks, Expert Gate based on autoencoders manages to assign test samples to the relevant tasks

equally accurately as a discriminative classifier (which assumes all training data is available).

- Task relatedness analysis
    - Expert Gate succeeds in predicting when a task could help another in the LwF framework and when it cannot (LwF vs fine-tuning decision).

My takeaways:

- This is an interesting point to take note of when thinking of a problem related to fine-tuning, especially when fine-tuning MoE.
    - Task biases when fine-tuning which can lead to suboptimal local minima.
        - If a model is trained on a task and fine-tuned on a widely different task, it can lead to suboptimal results due to the biases inferred in the initial task being different (think that the pre-training distribution shift can lead to local minima that is optimal for that distribution, but distribution of new tasks can be different and gain from other local minima that are unreachable due to the pre-training local minima – imagine the gradient descent valley)
- Expert Gate seems like DEMix.
    - Expert Gate focuses on the LwF or fine-tuning decision when being presented a new task, DEMix focuses more on the modularity of each expert.

    - Expert Gate focuses on task-level experts while DEMix focuses on domain-level experts.
    - Expert Gate experiments on computer vision tasks while DEMix focuses on NLP tasks.
- Both LwF and fine-tuning lead to the existing expert that was further trained with LwF or fine-tuning remaining unchanged while also creating a new expert. So, 2 experts are a result of this process (one old, one new).
- Both the routing to determine the similarity of an input with the tasks reflected in the existing experts and the task relatedness are determined by an autoencoder mechanism which is independent for each expert (it is trained as the expert is trained).
- The LwF method seems to be fine-tuning with a twist – instead of only fine-tuning with hard targets from the new data, fine-tune is done by considering the new data and soft targets given by the existing expert.
- Run through the methodology:
    - This method is a task-level MoE – it has the advantage of only routing the input sequence once. Since this is done at the beginning of inference, the selected task experts can be pre-loaded to memory and the routing does not need to be performed again, saving on memory costs of loading different experts for every new token.
    - Each task expert consists of the expert itself and an autoencoder, which is used for two things:

        - Determine the similarity of an input sequence to the task (how well does the task expert fit into the input sequence).
        - Determine the task relatedness between different tasks to help training of new experts.
    - Training new experts can be done in one of two ways:
        - LwF, which uses soft targets of the existing/old model to train a new model based on the new task's data.
        - Fine-tuning, which fine-tunes an existing/old expert with new data, resulting in a new expert.
    - Expert Gate also has the advantage of not all data needing to be stored on the same place at once for training. Since training can be done sequentially, training data can be used and sequentially discarded, saving on storage costs.
- The autoencoder is simply a function that deconstructs and attempts to reconstruct the input. The logic is that the closer the input is to the training data used to train that task's expert, the better the autoencoder will be at reconstructing the input.
- In computing the relatedness between two tasks, how can the reconstruction error of the autoencoder for task k on its own data be computed before the expert (and thus its autoencoder) is trained, since its initialization method relies on this task relatedness computation? This seems redundant.