Week 4 Tuesday Paper 1: Transfer Learning

Explained by Author's Presentation

Supplemental:

A Comprehensive Hands-on Guide to Transfer Learning with Real-World Applications in Deep Learning

1. What are the different options for (partially) transferring the information used in one vision network to another? What are the advantages and disadvantages of each? What else can we do? What is better or worse?

Ans:
Transferring without fine-tuning: Features are directly copied from one network to another without any further adjustments. This approach can quickly transfer knowledge but may lead to suboptimal performance on the new task due to the specificity of the higher layer features to the original task and optimization difficulties related to splitting networks in the middle of co-adapted layers.

Advantages: Quick to implement, leverages pre-learned features, and can be effective when the tasks are similar.

Disadvantages: May lead to suboptimal performance due to the specificity of features and optimization difficulties. The performance can degrade significantly when the tasks are very different or when the transferred features are from the higher layers, which are more specialized.

Transferring with fine-tuning: After copying features from one network to another, the features are fine-tuned on the new task. This approach can mitigate some of the drawbacks of transferring without fine-tuning by adjusting the copied features to better suit the new task.

Advantages: Allows the transferred features to adapt to the new task, potentially leading to better performance than transferring without fine-tuning, especially when the base and target tasks are not very similar.

Disadvantages: Requires additional computational resources for fine-tuning. There's a risk of overfitting, especially when the target dataset is small.

2. What are *selffer* networks and *transfer* networks? How do they serve the purpose of the experiment setup?

Ans: Selffer networks are controls where layers from a base network are copied and frozen, with higher layers trained on the same dataset. Transfer networks involve copying layers from one base network to train on a different dataset. These setups help understand the transferability and generality of features across tasks.

3. What architectures did the user build for the experiment? Do you think we would observe similar results for other more recent (as this is a 2014 paper) architectures?

Ans: The experiments utilized convolutional neural networks with eight layers, trained on ImageNet subsets. Similar results might be observed with more recent architectures, but the extent of transferability and the impact of co-adaptation could vary due to architectural differences.

4. What is *fragile co-adaptation*? How would you interpret the phenomena with the experiment results?

Ans: Fragile co-adaptation refers to the phenomenon where neurons across layers become interdependent, optimizing their functions collectively. This makes it difficult to transfer subsets of layers without losing performance, as the co-adapted features may not function optimally in isolation or in a different task setup.

With the experiment results, fragile co-adaptation can be interpreted as a key factor contributing to the degradation of performance when transferring features, especially from middle layers of the network. This indicates that neurons in these layers have developed interdependencies that are disrupted when the network is split, leading to a drop in performance. The experiments also demonstrate that while fine-tuning can recover some of the lost performance by re-establishing co-adapted interactions, the degree of transferability and the effectiveness of fine-tuning vary across layers, with early layers being more general and thus more easily transferable than higher, more specific layers.

5. This paper primarily focuses on Image tasks. What could be a problem if we apply the same method to natural language models? Can you think of a way to solve the problems?

Ans: Applying the same method to natural language models may face challenges due to the different nature of data and feature representation. Language models might require different strategies for transferring knowledge due to the sequential and contextual nature of text. Solving these problems could involve techniques like contextual embedding adaptation or cross-lingual transfer learning, considering the context-specific and syntax-driven features of language.

Week 4 Tuesday Paper 2: LoRA

Explained by Author

Supplemental:

- LoRA Fine-tuning & Hyperparameters Explained (in Plain English) | Entry Point AI
- **More Fine-tune methods:** https://docs.adapterhub.ml/methods.html
- **Blog:** https://www.databricks.com/blog/efficient-fine-tuning-lora-guide-llms

1. Before starting LoRA, read section 1 and 2 of the Adapter paper. The paper introduces *feature-based transfer*, *fine-tuning,* and *adapter modules*. In one sentence, what is each, and in one sentence for each, when is it likely to be good or bad?

Ans: Feature-based transfer is extracting and utilizing features from a pre-trained model in a new model, which is beneficial for leveraging learned representations but may lack flexibility for new tasks. Fine-tuning adjusts all or a significant portion of a pre-trained model's parameters on a new task, offering a good balance between leveraging pre-learned patterns and adapting to new tasks, but can be computationally expensive and risks overfitting. Adapter modules introduce small, task-specific modules within a pre-trained model, allowing for efficient task adaptation with minimal additional parameters, which is efficient and preserves the original model's integrity but might introduce some performance trade-offs compared to full fine-tuning.

2. Does LoRA change your answer to (1)? Explain briefly.

Ans: LoRA modifies this perspective by introducing a method that adapts large language models with minimal additional parameters without significantly increasing computational costs or altering the original model's structure. It suggests a balance between the flexibility of fine-tuning and the efficiency of adapters, potentially offering a new path for efficient model adaptation without the drawbacks of increased parameter count or computational demands.

3. Explain in plain language what equation 3 in the paper implies. What is inference latency? How does LoRA avoid it?

Ans: Equation 3 in the LoRA paper describes how the model updates its weights using low-rank matrices to adapt to new tasks without fully re-training the model. Inference latency is the time it takes or the delay during model prediction, critical in real-time

applications. LoRA minimizes inference latency by only modifying a small subset of parameters through low-rank approximations, thus avoiding the need for extensive computation that would delay predictions.

4. What is the time and space cost of *training* LoRA? What is the time and space cost of *running* LoRA once it is trained?

Ans: LoRA significantly reduces the VRAM usage during training, for instance, from 1.2TB to 350GB on GPT-3 175B. This is achieved by not needing to store the optimizer states for the majority of parameters that are frozen during the adaptation process. The hardware requirements are also reduced by up to three times, making the training process more efficient and lowering the barrier to entry.

The additional space required for LoRA is due to the low-rank matrices A and B introduced for each adapted layer. If r is the rank and d is the dimension of the weight matrix being adapted, the space complexity for each layer is $O(dr+rd)=O(2dr)$. Considering the adaptation of multiple layers, if L layers are adapted, the total space complexity becomes $O(L \cdot 2dr)$.
The time cost of training with LoRA involves the computation required to apply the low-rank updates during forward and backward propagation. This includes the multiplication of the low-rank matrices with the input and gradient, respectively. The complexity of matrix multiplication for matrices of size $d \times r$ and $r \times d$ is $O(d \cdot r \cdot d)$ for each layer. If L layers are adapted, and considering the need to compute both forward and backward passes, the total time complexity for the training process can be approximated as $O(L \cdot d.r.d)$.

After training, the space complexity remains $O(L \cdot 2dr)$ due to the additional low-rank matrices that are stored along with the original model's parameters. This is a relatively small increase in the overall model size, especially since $r \ll d$ and the time complexity also remains the same

5. What hyperparameters must be set to use LoRA? What are some basic intuitions behind choosing appropriate values for these hyperparameters?

Ans: To use LoRA, one must choose hyperparameters like the rank of the adaptation matrices and the specific layers to apply these adaptations. The basic intuition behind selecting these values involves balancing the model's adaptability with maintaining computational efficiency. A lower rank might suffice for significant performance gains

with minimal computational overhead, while selecting which layers to adapt depends on their relevance to the task at hand.