ECE472 – Quiz 11

Jonathan Lam

December 8, 2020

NeurIPS 2020: "Towards Neural Programming Interfaces"

I glanced over three or four other randomly-chosen papers before settling on this one. This one definitely caught my eye because of its relationship to APIs, and it far surpassed my expectation in its attempt to truly create a neural-network based interface ("neural programming interface," or NPI). Not only that, but it's a wonderful agglomeration of some of the later topical methods we covered in class, e.g., text-generation models, adversarial models, and style-transfer (the latter two are used in a physically similar but functionally very different way). It was also interesting that this paper sprung up concurrently with and independently of a different method to control the output of neural networks (PPLM), and the authors describe how their NPI performs favorably and doesn't require pre-labeled training data.

The general idea of this paper is that, given some pretrained neural network (e.g., some large generative model based on a transformer architecture), we can train an auxiliary, separate neural network to bias the outputs of the pre-trained one. It's called an "NPI" due to its similarity to an API: this NPI can pass high-level commands to the mostly-black-box-of-the-larger-neural-network to retrieve an output that was modified by those commands. Example commands for a text-generative network like GPT-2 are to avoid using a certain word/phrase (e.g., offensive language filtering) or topic control. A very general use of this is to undo bias: if bias is found in the network's generated text, then we can apply a command that reverses that bias.

At a very high-level, the NPI is trained to generate a certain "command" by recording some of the activations of the main network when inputs are fed to the main network. The corresponding outputs are analyzed with respect to their similarity to the desired command (by a loss related to the style-transfer loss), and labeled accordingly. This auto-labeling allows us to train for the best set of activation values to produce the desired command. The NPI is used by adding those recorded activation values onto the equivalent activations during inference time to bias the main network to produce similar outputs. Lastly, the NPI is trained adversarially against a discriminator and a content-classifier.

This is comparable to fine-tuning on a specific domain, but the authors note that NPIs overcome two shortcomings of fine-tuning: "catastrophic forgetting" of the language-model and more fine-tuned "content control" (e.g., filtering at

the granularity of words and phrases rather than subject matter).

A question I might ask someone else about this paper is: *What does this model relate to the trend of model biggening (e.g., GPT-2 vs. GPT-3)? What are some implications?*

I ask this because the intent of the paper, as well as the methodologies involved, deal with specific types of architectures, e.g., the super-large architectures such as GPT-{1,2,3}, MEENA, GeDi, and similar text- or speech-generating neural networks, all of which are very large networks. In particular, the authors focus most of their tests on GPT-2, and attempt to manipulate it to produce goal-oriented tasks, and their results show a fair degree of success.

This begs the question: why not just train a larger network? With few changes to GPT-2 other than increasing the number of parameters, GPT-3 has shown wonders in zero-, one-, or few-shot learning (i.e., no fine-tuning) in different tasks. The authors address this briefly by saying that GPT-3 is a predictive model rather than a goal-oriented text generative model. I'm not entirely convinced that this is a strong distinction.

I believe that this discourse between larger and smaller networks, and the trade-offs between fine-tuning, few-shot learning, and NPIs is interesting. On a (relatively) smaller generative network, we can fine-tune to specific tasks to improve domain-specific knowledge. On a larger network, we can miss out on a lot of the domain-specific training and still get high accuracy in the same tasks. But NPIs encourage some flexibility on both ends of the spectrum: we can use this, rather than fine-tuning, on a smaller network to achieve comparable performance on domain-specific tasks (without the same pitfalls of fine-tuning). On a larger network, perhaps this can achieve better granularity in domain-specific tasks than few-shot learning.

In both cases, the analogy to an API is very apt: we can use these NPI "controllers" to give us the output we need by providing specific commands. Just like an API, this may be a useful step in making large models more available to researchers in their particular applications (with specific commands). If this readily outperforms fine-tuning, perhaps this will encourage more medium-sized models that can be programmed to perform a variety of tasks more effectively than before, which may be useful to researchers who do not have the ability to train the larger models.