Model Persistence

CSCI 4850/5850 - Neural Networks

Auxiliary materials

Motivation:

I got a question about using/deploying models last week and thought I would post some materials here for those interested. We will not be using any of this in the course, but there is a simple (generic) way to do this (which I provide below). There are also much more complex and platform-specific, but inevitably more practical, ways to do this also (for which I provide existing tutorials/documentation below). Learning the generic way will get you started on the right path, and then this will help you understand how to follow-up with more platform-specific methods.

Generic Model Persistence

To keep it simple, I posted two code notebooks on the course website:

https://jupyterhub.cs.mtsu.edu/azuread/services/csci4850-materials/download.php?filename=ModelPersistence-Saving.html

The first one creates a model for the Iris data set, checks the validation accuracy, and saves the model using check-pointing (`trainer.save_checkpoint()`). The same is repeated for an encoder-decoder model, but that also needs to save the tokenizer components (in json format).

https://jupyterhub.cs.mtsu.edu/azuread/services/csci4850-materials/download.php?filename=ModelPersistence-Loading.html

This second one will load the models from the checkpoint files (`LightningModule.load_from_checkpoint()`) and then recalculate the validation accuracies to see that they agree with the previous notebook. Checkpoints basically just contain model hyperparameters and parameters (weights), so you still need the source code for the model classes to load a checkpoint.

Models are typically deployed in python by just loading them from checkpoints and then passing new data into them (they are just functions in the end that take tensors as input and produce tensors as output). This is pretty low-level and requires you to write all of the code needed to preprocess and postprocess model inputs/outputs, but is also a comprehensive method that can work in any situation.

Advanced Training/Validation/Testing/Deployment

For more advanced deployment, folks will create artifacts (like model templates/code, checkpoint files, datasets, tokenizer json files, etc.) and push them onto a cloud service that can allow applications to easily pull/load/use the model. HuggingFace and Weights and Biases are two such platforms, but there are several other providers (I am partial to WandB, myself, but HF is very popular right now). This approach requires some engineering though since these artifacts need to conform to certain standards set by the platform. Weights and Biases is a little more flexible in this regard, but Transformers (the library used to connect with HuggingFace) is a lot more opinionated on how things need to be wrapped/formatted. However, there are some advantages to matching standards that many people are using - it's easy for others to adapt their code to work with or use your pretrained model. WandB assumes the user is a little more informed about deep learning models/training, but therefore potential collaborators may need to write more custom code for their use-cases.

HuggingFace provides tutorials, but it does have many steps:
https://huggingface.co/docs/transformers/en/create_a_model

There are similar instructions for WandB: https://docs.wandb.ai/tutorials/artifacts

WandB even has natural hooks for working with PyTorch Lightning:
https://docs.wandb.ai/tutorials/lightning

Since this process is more about software engineering at that point and less about neural networks themselves, I don't cover it much in the course, but I hope this connects the dots for some folks.