Published in Towards Data Science · Follow

Manu Joseph · Follow
Jan 27, 2021 · 6 min read

PyTorch Tabular — A Framework for Deep Learning for Tabular Data

It is common knowledge that Gradient Boosting models, more often than not, kick the asses of every other machine learning models when it comes to Tabular Data. I have written extensively about Gradient Boosting, the theory behind and covered the different implementations like XGBoost, LightGBM, CatBoost, NGBoost etc. in detail.

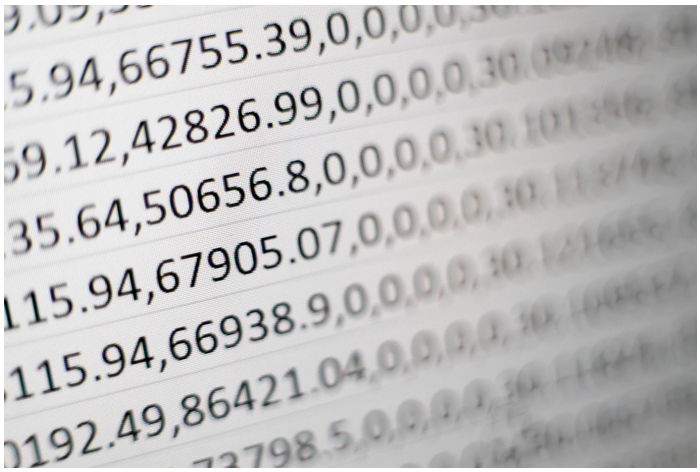

Photo by Mika Baumeister on Unsplash

The unreasonable effectiveness of Deep Learning that was displayed in many other modalities — like text and image- haven not been demonstrated in tabular data. But lately, the deep learning revolution have shifted a little bit of focus to the tabular world and as a result, we are seeing new architectures and models which was designed specifically for tabular data modality. And many of them are coming up as an equivalent or even slightly better than well-tuned Gradient Boosting models.

What is PyTorch Tabular?

PyTorch Tabular

PyTorch Tabular is a framework/ wrapper library which aims to make Deep Learning with Tabular data easy and accessible to real-world cases and research alike. The core principles behind the design of the library are:

Instead of starting from scratch, the framework has been built on the shoulders of giants like **PyTorch**(obviously), and **PyTorch Lightning**.

It also comes with state-of-the-art deep learning models that can be easily trained using pandas dataframes.

- The high-level config driven API makes it very quick to use and iterate. You can just use a pandas dataframe and all of the heavy lifting for normalizing,

packaged with the library.

- State-of-the-art networks like Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data, and TabNet: Attentive Interpretable Tabular Learning are implemented. See examples from the documentation for how to use them.
- By using PyTorch Lightning for the training, PyTorch Tabular inherits the flexibility and scalability that Pytorch Lightning provides

Why PyTorch Tabular?

PyTorch Tabular aims to reduce the barrier for entry for both industry application and research of Deep Learning for Tabular data. As things stand now, working with Neural Networks is not that easy; at least not as easy as traditional ML models with Sci-kit Learn.

PyTorch Tabular attempts to make the "software engineering" part of working with Neural Networks as easy and effortless as possible and let you focus on the model. I also hopes to unify the different developments in the Tabular space into a single framework with an API that will work with different state-of-the-art models.

Right now, most of the developments in Tabular Deep Learning are scattered in individual Github repos. And apart from (which I love and hate), no framework has really paid attention to Tabular Data. And this is the need which led to PyTorch Tabular.

How to use PyTorch Tabular?

Installation

First things first — let's look at how we can install the library.

Although the installation includes PyTorch, the best and recommended way is to first install PyTorch from here, picking up the right CUDA version for your machine. (PyTorch Version >1.3)

Once, you have got Pytorch installed, just use:

```
pip install pytorch_tabular[all]
```

to install the complete library with extra dependencies(Weights & Biases for Experiment Tracking).

And :

```
pip install pytorch_tabular
```

for the bare essentials.

The sources for pytorch_tabular can be downloaded from the `Github repo`.

You can either clone the public repository:

```
git clone git://github.com/manujosephv/pytorch_tabular
```

Once you have a copy of the source, you can install it with:

Setting up the Configs

There are four configs that you need to provide(most of them have intelligent default values), which will drive the rest of the process.

- **DataConfig** — Define the target column names, categorical and numerical column names, any transformation you need to do, etc.
- **ModelConfig** — There is a specific config for each of the models. This determines which model we are going to train and also lets you define the hyperparameters of the model
- **TrainerConfig** — This let's you configure the training process by setting things like batch_size, epochs, early stopping, etc. The vast majority of parameters are directly borrowed from PyTorch Lightning and is passed to the underlying Trainer object during training
- **OptimizerConfig** — This let's you define and use different Optimizers and LearningRate Schedulers. Standard PyTorch Optimizers and Learning RateSchedulers are supported. For custom optimizers, you can use the parameter in the fit method to overwrite this. The custom optimizer should be PyTorch compatible
- **ExperimentConfig** — This is an optional parameter. If set, this defines the Experiment Tracking. Right now, only two experiment tracking frameworks are supported: Tensorboard and Weights&Biases. W&B experiment tracker has more features like tracking the gradients and logits across epochs.

```
data_config = DataConfig( target=['target'],
continuous_cols=num_col_names, categorical_cols=cat_col_names, )

trainer_config = TrainerConfig( auto_lr_find=True, batch_size=1024,
max_epochs=100, gpus=1)

ptimizer_config = OptimizerConfig()

model_config = CategoryEmbeddingModelConfig( task="classification",
layers="1024-512-512", # Number of nodes in each layer
activation="LeakyReLU", # Activation between each layers learning_rate
= 1e-3 )
```

Initializing the Model and Training

Now that we have the configs defined, we need to initialize the model using these configs and call the *fit* method.

```
tabular_model = TabularModel( data_config=data_config,
model_config=model_config, optimizer_config=optimizer_config,
trainer_config=trainer_config, ) tabular_model.fit(train=train,
validation=val)
```

That's it. The model will be trained for the specified number of epochs.

Model List

- FeedForward Network with Category Embedding is a simple FF network, but with and Embedding layers for the categorical columns. This is very similar to the *fastai* Tabular Model
- Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data is a model presented in ICLR 2020 and according to the authors have beaten well-tuned Gradient Boosting models on many datasets.
- TabNet: Attentive Interpretable Tabular Learning is another model coming out of Google Research which uses Sparse Attention in multiple steps of decision making to model the output.

To implement new models, see the How to implement new models tutorial. It covers basic as well as advanced architectures.

during training, we can use the method.

```
result = tabular_model.evaluate(test)
```

```
--------------------------------------------------------------------------
---------- DATALOADER:0 TEST RESULTS
{
'test_accuracy': tensor(0.6924, device='cuda:0'),
'train_accuracy': tensor(0.6051, device='cuda:0'),
'train_loss': tensor(0.6258, device='cuda:0'),
'valid_accuracy': tensor(0.7440, device='cuda:0'),
'valid_loss': tensor(0.5769, device='cuda:0')}
--------------------------------------------------------------------------
----------
```

Getting Predictions on Unseen Data

To get the prediction as a dataframe, we can use the `predict` method. This will add predictions to the same dataframe that was passed in. For classification problems, we get both the probabilities and the final prediction taking 0.5 as the threshold

```
pred_df = tabular_model.predict(test)
```

Saving and Loading Models

We can also save a model and load it later for inferencing.

```
tabular_model.save_model("examples/basic")

loaded_model = TabularModel.load_from_checkpoint("examples/basic")

result = loaded_model.evaluate(test)
```

Code, Documentation and How to Contribute

The code for the framework is available at PyTorch Tabular: A standard framework for modelling Deep Learning Models for tabular data (github.com).

Documentation and tutorials can be found at PyTorch Tabular (pytorch-tabular.readthedocs.io)

Contributions are more than welcome and details about how to contribute is also laid out here.

Related Work

fastai is the closest to PyTorch Tabular, both built on PyTorch. But where PyTorch Tabular differentiates from fastai is with it's modular and decoupled nature and it's usage of standard PyTorch and PyTorch Lightning components which makes adoption, including new models, and hacking the code much more easy than with *fastai*.

References

[1] Sergei Popov, Stanislav Morozov, Artem Babenko. *"Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data"*. arXiv:1909.06312 [cs.LG] (2019)

[2] Sercan O. Arik, Tomas Pfister;. *"TabNet: Attentive Interpretable Tabular Learning"*. arXiv:1908.07442 (2019).

What Next?